

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

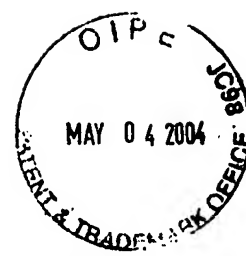
Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**

Docket No.: 56937-100



PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of	:	Customer Number: 20277
Kei YONEDA, et al.	:	Confirmation Number: 4830
Serial No.: 10/727,055	:	Group Art Unit: 2122
Filed: December 04, 2003	:	Examiner:
For: SOFTWARE PROCESSING METHOD AND SOFTWARE PROCESSING SYSTEM	:	

TRANSMITTAL OF CERTIFIED PRIORITY DOCUMENT(S)

Mail Stop CPD
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

At the time the above application was filed, priority was claimed based on the following application:

Japanese Patent Application No. JP 2002-355683, filed on December 6, 2002.

A copy of each priority application listed above is enclosed.

----- Respectfully submitted, -----

MCDERMOTT, WILL & EMERY


Michael E. Fogarty
Registration No. 36,139

600 13th Street, N.W.
Washington, DC 20005-3096
(202) 756-8000 MEF:gav
Facsimile: (202) 756-8087
Date: May 4, 2004

10/727,055

December 4, 2003

Kei Yano et al.

日本国特許庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されて
いる事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed
with this Office.

出願年月日

Date of Application:

2002年12月 6日

出願番号

Application Number:

特願2002-355683

[ST. 10/C]:

[JP 2002-355683]

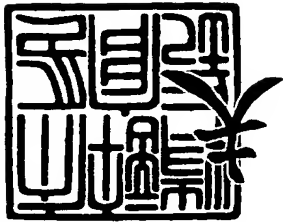
出願人

Applicant(s):

松下電器産業株式会社

2003年11月17日

今井 康



特許庁長官
Commissioner,
Japan Patent Office

出証番号 出証特2003-3094778

【書類名】 特許願

【整理番号】 5037740044

【あて先】 特許庁長官 殿

【国際特許分類】 G06F 13/00

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 米田 圭

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 河本 功

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 喜田 誠司

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 松林 貴明

【特許出願人】

【識別番号】 000005821

【氏名又は名称】 松下電器産業株式会社

【代理人】

【識別番号】 100086737

【弁理士】

【氏名又は名称】 岡田 和秀

【電話番号】 06-6376-0857

【手数料の表示】

【予納台帳番号】 007401

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 9305280

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 ソフトウェア処理方法およびソフトウェア処理システム

【特許請求の範囲】

【請求項 1】 プロセッサが使用するリソースの使用状況を監視する使用状況監視処理の過程と、

前記使用状況監視処理の過程で得られる競合情報に応じて、実行されるソフトウェアの処理方法を変更するソフトウェア処理変更処理の過程とを含むことを特徴とするソフトウェア処理方法。

【請求項 2】 前記ソフトウェア処理変更処理の過程は、ソフトウェアがある処理を行うのに複数の実行方法を持ち、前記ソフトウェアの実行中に前記使用状況監視処理の過程で得られる競合情報に応じて、前記複数の実行方法の中から 1 つを選択することを特徴とする請求項 1 に記載のソフトウェア処理方法。

【請求項 3】 前記リソースは、処理の記憶装置および前記プロセッサと前記記憶装置を接続するバスであり、

前記使用状況監視処理の過程は、前記バスの使用状況を監視することを特徴とする請求項 1 または請求項 2 に記載のソフトウェア処理方法。

【請求項 4】 前記使用状況監視処理の過程は、複数クロック遡ったバスの使用状況を記憶し、過去および現在の使用状況から前記競合情報を生成することを特徴とする請求項 3 に記載のソフトウェア処理方法。

【請求項 5】 前記使用状況監視処理の過程は、前記バスの使用時における使用時間を記憶し、前記使用時間が所定値以上か否かに基づき前記競合情報を生成することを特徴とする請求項 3 に記載のソフトウェア処理方法。

【請求項 6】 前記リソースは、前記プロセッサからの処理要求に応じて処理を行う第 2 のプロセッサであって、

前記使用状況監視処理の過程は、前記第 2 のプロセッサの使用状況の監視を行うことを特徴とする請求項 1 に記載のソフトウェア処理方法。

【請求項 7】 請求項 6 に記載のソフトウェア処理方法が、さらに、同一アドレスでアクセス可能な複数のメモリバンクを有し、前記使用状況監視処理の過程で得られる競合情報は、前記複数のメモリバンクのうち 1 つのメモリバンクの選

択を示す信号であることを特徴とする請求項 6 に記載のソフトウェア処理方法。

【請求項 8】 請求項 1 に記載のソフトウェア処理方法がさらにソフトウェアのコンパイラを有し、

前記コンパイラは、

ソフトウェアから前記リソースを使用する処理を識別する処理識別手段と、

前記処理識別手段により識別された処理と等価な、前記リソースを使用しない等価処理と、

前記使用状況監視処理の過程で得られる競合情報により使用状況を判定する使用状況判定処理と、

前記使用状況判定処理の結果に応じて、前記処理識別手段により識別された処理を前記等価処理に置き換える置換処理とを、前記ソフトウェアに追加することを特徴とする請求項 1 に記載のソフトウェア処理方法。

【請求項 9】 請求項 1 に記載のソフトウェア処理方法がさらにソフトウェアのコンパイラを有し、

前記コンパイラは、

ソフトウェアから前記リソースを使用する処理を識別する処理識別手段と、

前記処理識別手段により識別された処理と等価な、前記リソースを使用しない等価処理と、

現時刻の前記使用状況監視処理の過程で得られる競合情報を記憶する記憶処理と、

過去の時刻における前記競合情報により使用状況を判定する使用状況判定処理と、

前記使用状況判定処理の結果に応じて、前記処理識別手段により識別された処理を前記等価処理に置き換える置換処理とを、前記ソフトウェアに追加することを特徴とする請求項 1 に記載のソフトウェア処理方法。

【請求項 10】 前記競合情報は、前記リソースの処理要求が発行されてから処理が終了するまでの処理時間であり、

前記使用状況判定処理は、前記処理時間がある設定値と比較する処理であることを特徴とする請求項 8 または請求項 9 に記載のソフトウェア処理方法。

【請求項 1 1】 前記競合情報は、前記リソースの処理要求が発行されてから処理が実行するまでの待ち時間であり、

前記使用状況判定処理は、前記待ち時間がある設定値と比較する処理であることを特徴とする請求項 8 または請求項 9 に記載のソフトウェア処理方法。

【請求項 1 2】 前記使用状況判定処理は、定期的または不定期に前記リソースの使用状況の判定を見直すことを特徴とする請求項 8 から請求項 1 1 までのいずれかに記載のソフトウェア処理方法。

【請求項 1 3】 前記使用状況判定処理は、乱数を用いて前記リソースの使用状況の判定を見直すことを特徴とする請求項 1 2 にソフトウェア処理方法。

【請求項 1 4】 請求項 9 に記載のソフトウェア処理方法において、
前記コンパイラは、前記処理識別手段により抽出される処理が前記ソフトウェアの複数の箇所から抽出される場合、さらに、前記処理識別手段で識別された処理の出現箇所を識別する出現箇所識別処理を前記ソフトウェアに追加し、前記記憶処理は前記出現箇所毎に前記競合情報を記憶し、前記使用状況判定処理は前記出現箇所毎に記憶した前記競合情報を用いて判定を行うことを特徴とする請求項 9 に記載のソフトウェア処理方法。

【請求項 1 5】 プロセッサと、
前記プロセッサが使用するリソースと、
前記リソースの使用状況を監視する使用状況監視装置と、
前記使用状況監視装置で得られる競合情報に応じて、実行されるソフトウェアの処理方法を変更するソフトウェア処理変更装置を有することを特徴とするソフトウェア処理システム。

【請求項 1 6】 前記ソフトウェア処理変更装置は、ソフトウェアがある処理を行うのに複数の実行方法を持ち、前記ソフトウェアの実行中に前記競合情報に応じて、前記複数の実行方法の中から 1 つを選択することを特徴とする請求項 1 5 に記載のソフトウェア処理システム。

【請求項 1 7】 前記リソースは、前記プロセッサからの処理要求に応じて処理を行う第 2 のプロセッサであって、

前記使用状況監視装置は、前記第 2 のプロセッサの使用状況の監視を行うこと

を特徴とする請求項 15 に記載のソフトウェア処理システム。

【請求項 18】 前記競合情報は、前記プロセッサへの割り込み信号であることを特徴とする請求項 17 に記載のソフトウェア処理システム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、プロセッサと共有リソースを有するシステムにかかわり、特に、共有リソース競合時の処理速度の低下を防止する技術に関するものである。

【0002】

【従来の技術】

共有リソースを有するシステムとしては様々なものがある。マルチプロセッサシステム、例えば CPU のようなプロセッサと DSP (デジタルシグナルプロセッサ) で構成されるマルチプロセッサシステムにおいては、処理をプロセッサと DSP で分担して実行することにより処理性能の向上を図りながら、メモリ、バス、周辺回路などを共有リソースとして共用することにより、チップ面積を縮小し、低コストを実現している。共有リソースとしては、通常、メモリ、バス、周辺回路などがあり、システムの目的、仕様に応じて様々である。

【0003】

共有リソースに対してアクセス競合が発生した場合、例えば、プロセッサが外部メモリへのデータ読み出し中に、DSP から同一のメモリに対してデータ読み出しの処理が発行された場合、バス調停回路によりアクセス競合に対する処理が制御される。バス調停回路の制御方法としては、DSP からのデータ読み出し処理は受け付けるが、プロセッサからのデータ読み出しが終了するまでは DSP からのデータ読み出しの処理は待たせる、あるいは実行中のプロセッサからのデータ読み出しをキャンセルし、DSP からのデータ読み出しを優先に実行し、DSP での処理の終了後、プロセッサからのデータ読み出し処理を再実行する、といった具合に、処理の仕様に応じて様々である。

【0004】

さらに、プロセッサと DSP で構成されるマルチプロセッサシステムにおいて

、DSPはプロセッサからの処理要求を受け付けることにより処理を実行する場合、プロセッサとDSPとの間の制御方式が突き放し制御方式であれば、DSPを共有リソースとすることができる。突き放し制御方式におけるDSPは、プロセッサからの処理要求をDSPの状態に関係なく受け付けるが、処理の実行に関してはDSPの状態を判断した後で開始する。つまり、DSPが別の処理を実行中であれば、処理要求を受け付けてもすぐには処理を開始させずにしばらく待たせ、DSP側でプロセッサからの処理要求が実行できる状態になった時点で自律的に処理を実行し、処理結果をプロセッサ側へ返す。この制御方式では、DSPまたは別回路などの調停回路においてプロセッサが発行する処理をスケジューリングすることにより、複数の処理が共通のDSPを競合することなく使用することになる。したがって、この場合、DSPは共有リソースであるといえることができる。

【0005】

【非特許文献1】

斎藤忠夫・発田弘著、「高性能コンピュータアーキテクチャ」、丸善株式会社、p290、p309

【0006】

【発明が解決しようとする課題】

しかしながら、従来の共有リソースを有するシステムでは、共有リソースへのアクセス競合が発生した場合、調停回路により競合リソースを使用する処理を適当にスケジューリングするが、優先順位の低い処理は先の処理が終了するまで待つことになり、共有リソースへのアクセス競合が頻繁に発生するような場合、システム全体の処理速度が低下する要因となる。

【0007】

さらに、調停回路は共有リソースの状態のみを把握しており、システム全体の状態を把握していない。例えばDSPが他の処理を実行中にプロセッサからDSPへの処理要求が発行された場合、従来の調停回路では、先の処理が終了するまでその処理を待たせる。一方、プロセッサはDSPへ発行した処理の結果を受けて次の処理を実行するため、プロセッサも待ち状態となる場合がある。このよう

な状況では、先の処理が終了するまでDSPへ発行された処理およびその発行元であるプロセッサが待ち状態となり、プロセッサおよびDSPを効率良く使用することができない。

【0008】

また、上記従来の共有リソースを有するシステムでは、DSPへの処理要求が重複するDSP競合を避けるためにソフトウェアをCPUとDSPに適切に割り振る必要がある。この割り振り作業は人手で行っており、ソフトウェアの修正が発生する毎に多大な時間を必要とし、工数がかかるという問題がある。

【0009】

【課題を解決するための手段】

上記の課題を解決するために、本発明は次のような手段を講じる。

【0010】

本発明によるソフトウェア処理方法は、プロセッサとプロセッサが使用するリソースで構成されるシステムに対し、プロセッサが使用するリソースの使用状況を監視する使用状況監視処理の過程と、前記使用状況監視処理の過程で得られる競合情報に応じて、実行されるソフトウェアの処理方法を変更するソフトウェア処理変更処理の過程とを含むものである。

【0011】

上記のソフトウェア処理方法に対応する本発明によるソフトウェア処理システムは、プロセッサと、前記プロセッサが使用するリソースと、前記リソースの使用状況を監視する使用状況監視装置と、前記使用状況監視装置で得られる競合情報に応じて、実行されるソフトウェアの処理方法を変更するソフトウェア処理変更装置を有するものである。

【0012】

上記構成によれば、リソースの競合を動的に判定し、リソースを使用するソフトウェアの処理方法を変更することで、リソース競合時の処理の停止を避け、システムの処理速度の低下を抑制することができる。

【0013】

上記において、前記ソフトウェア処理変更処理の過程、または、前記ソフトウ

エア処理変更装置についての1つの態様は、ソフトウェアがある処理を行うのに複数の実行方法を持ち、前記ソフトウェアの実行中に前記使用状況監視処理の過程で得られる競合情報に応じて、前記複数の実行方法の中から1つを選択するものである。

【0014】

上記において、前記リソースが、処理の記憶装置および前記プロセッサと前記記憶装置を接続するバスである場合において、前記使用状況監視処理の過程についての1つの態様は、前記バスの使用状況を監視するものである。これによれば、バスアクセス競合の発生を軽減し、バスを効率良く使用することが可能となる。したがって、バスアクセスの競合の発生によるシステムの処理速度の低下の影響を軽減することができる。

【0015】

上記において、前記使用状況監視処理の過程についての1つの態様は、複数クロック遡ったバスの使用状況を記憶し、過去および現在の使用状況から前記競合情報を生成することである。また、上記において、前記使用状況監視処理の過程についての別の態様は、前記バスの使用時における使用時間を記憶し、前記使用時間が所定値以上か否かに基づき前記競合情報を生成することである。クロック数または時間を指標にして、バスの使用状況を容易に判定することができる。

【0016】

上記において、前記リソースが、前記プロセッサからの処理要求に応じて処理を行う第2のプロセッサである場合において、前記使用状況監視処理の過程、または、前記使用状況監視装置についての1つの態様は、前記第2のプロセッサの使用状況の監視を行うことである。この場合、第2のプロセッサを使用しているときに別の処理が発生した場合、前記競合情報として前記プロセッサへの割り込み信号を用いて、割り込みを発生させる割り込み機能を備えたとよい。

【0017】

上記のソフトウェア処理方法が、さらに、同一アドレスでアクセス可能な複数のメモリバンクを有する場合において、前記使用状況監視処理の過程で得られる競合情報についての1つの態様は、前記複数のメモリバンクのうち1つのメモリ

バンクの選択を示す信号である。この場合、第2のプロセッサでの処理の実行毎に、あるいは第2のプロセッサで実行された処理が終了する毎に、複数のメモリバンクの中から1つを選択する選択信号を切り替える機能を備えるとよい。

【0018】

上記によれば、プロセッサの競合を動的に判定し、割り込みを行ったり、メモリバンクを選択することで、プロセッサ競合時の処理の停止を避け、システムの処理速度の低下を抑制することができる。

【0019】

上記のソフトウェア処理方法が、さらにソフトウェアのコンパイラを有する場合において、前記コンパイラについての1つの態様は、ソフトウェアから前記リソースを使用する処理を識別する処理識別手段と、前記処理識別手段により識別された処理と等価な、前記リソースを使用しない等価処理と、前記使用状況監視処理の過程で得られる競合情報により使用状況を判定する使用状況判定処理と、前記使用状況判定処理の結果に応じて、前記処理識別手段により識別された処理を前記等価処理に置き換える置換処理とを、前記ソフトウェアに追加することである。これにより、リソース競合を避け、システムの処理速度の低下を避ける。

【0020】

また、上記のソフトウェア処理方法が、さらにソフトウェアのコンパイラを有する場合において、前記コンパイラについての1つの態様は、ソフトウェアから前記リソースを使用する処理を識別する処理識別手段と、前記処理識別手段により識別された処理と等価な、前記リソースを使用しない等価処理と、現時刻の前記使用状況監視処理の過程で得られる競合情報を記憶する記憶処理と、過去の時刻における前記競合情報により使用状況を判定する使用状況判定処理と、前記使用状況判定処理の結果に応じて、前記処理識別手段により識別された処理を前記等価処理に置き換える置換処理とを、前記ソフトウェアに追加することである。これによれば、競合の履歴を加味した処理となり、リソース競合の判定を高精度に行うことができる。

【0021】

上記において、前記競合情報についての1つの態様は、前記リソースの処理要

求が発行されてから処理が終了するまでの処理時間であり、この場合に、前記使用状況判定処理は、前記処理時間がある設定値と比較する処理を行うものである。また、上記において、前記競合情報についての別の態様は、前記リソースの処理要求が発行されてから処理が実行するまでの待ち時間であり、この場合に、前記使用状況判定処理は、前記待ち時間がある設定値と比較する処理を行うものである。これによれば、処理の待ち時間や処理時間などを計測することでリソース競合が発生中か否かを判定し、リソース競合が発生中ならば新たにリソース競合を引き起こす処理を発行しない。これにより、競合によるシステムの処理速度の低下を抑制することが可能である。

【 0 0 2 2 】

上記において、前記使用状況判定処理についての 1 つの態様は、定期的または不定期に前記リソースの使用状況の判定を見直すことである。この場合に、乱数を用いて前記リソースの使用状況の判定を見直すのがよい。これによれば、定期的または不定期に競合判定を補正するので、さらに高精度な競合判定が可能である。

【 0 0 2 3 】

上記のソフトウェア処理方法において、前記コンパイラにおける前記処理識別手段により抽出される処理が前記ソフトウェアの複数の箇所から抽出される場合、前記コンパイラについての 1 つの態様は、さらに、前記処理識別手段で識別された処理の出現箇所を識別する出現箇所識別処理を前記ソフトウェアに追加し、前記記憶処理は前記出現箇所毎に前記競合情報を記憶し、前記使用状況判定処理は前記出現箇所毎に記憶した前記競合情報を用いて判定を行うことである。これによれば、競合情報を競合を引き起こす処理の出現箇所とともに記憶することで、ループ処理における競合判定を高精度に行うことが可能である。

【 0 0 2 4 】

以上のようにして、本発明によれば、リソースの競合およびプロセッサ上で実行する処理の競合を考慮した人手での処理の割り当てを行うことがなくなり、工数の削減が図れる。

【 0 0 2 5 】

【発明の実施の形態】

図1は、本発明の実施の形態にかかわるソフトウェア処理システムの基本構成を示すブロック図である。

【0026】

図1において、プロセッサ101がソフトウェア処理の実行を始めると、使用状況監視処理手段103は、プロセッサ101が使用するリソース102の使用状況を監視し、その出力を受けて、リソース102の使用状況の情報を取得する。リソース102の使用状況の情報の結果に応じて、ソフトウェア処理変更処理手段104は実行されるソフトウェアの処理方法を変更する。

【0027】

なお、前記使用状況監視処理手段103および前記ソフトウェア処理変更処理手段104は、それぞれソフトウェアで実現したものを想定しているが、それぞれハードウェアで実現しても良く、少なくとも、リソースの使用状況を監視し、その情報を取得する機能、および、前記情報の出力を受けてソフトウェアの処理方法を変更する機能を有していれば良いものである。

【0028】

以下、本発明の第1から第6までの実施の形態について、図を用いて説明する。

【0029】

(第1の実施の形態)

以下、本発明の第1の実施の形態を図面に基づいて説明する。

【0030】

図2は、本発明の第1の実施の形態にかかわるソフトウェア処理システムのハードウェアの構成を示すブロック図である。

【0031】

図2において、1001はCPU、1002はCPU1001が使用するバス、1003はCPU1001の周辺回路、1004は外部メモリ、1005はバス1002上のバスアクセスの状態を記憶する使用状況監視装置である。

【0032】

図3は、本発明の第1の実施の形態にかかわるソフトウェア処理システムのバス1002の使用状況監視装置1005の構成を示す図である。

【0033】

図3において、1101はバス1002上でバスアクセスが起きているかどうかを特定の期間だけ識別するバスアクセス識別装置、1102はバスアクセス識別装置1101の出力を受けて、バスアクセスの状態を記録するバスアクセス状態レジスタである。

【0034】

図4は、本発明の第1の実施の形態にかかわるソフトウェア処理システムのソフトウェアの構成を示す図である。

【0035】

図4において、1201はCPU1001上で動作するプログラム、1202, 1203, 1204はプログラム1201中に記述される処理、1205, 1206, 1207は処理1202, 1203, 1204とそれぞれ同等の機能を持ち、バス1002を使用しないCPU1001用の処理である。

【0036】

図5は、本発明の第1の実施の形態にかかわるソフトウェア処理システムにおけるコンパイルフローを示すフローチャートである。

【0037】

図5において、1301はコンパイラがプログラム1201をコンパイルするコンパイル手段、1302はプログラム1201に記述される処理において、バス1002を使用する処理を識別する処理識別手段、1303は処理識別手段1302の出力を受けて、使用状況監視装置1005が持つバスアクセス状態レジスタ1102の値を読み取り、バスアクセスの情報を動的に判定するプログラムである使用状況判定処理、および、処理識別手段1302により識別された処理を、処理識別手段1302により識別された処理と等価な前記リソースを使用しない等価処理に置き換える置換処理を付加する処理付加手段である。

【0038】

図6は、本発明の第1の実施の形態にかかわるソフトウェア処理システムにお

けるバスアクセス識別装置 1101 が識別するバスアクセスの状態を示す図であり、図 7 は、そのソフトウェア処理システムにおける処理付加手段 1303 によるソフトウェアの変更を示す図である。

【0039】

以下、第 1 の実施の形態のソフトウェア処理システムの動作について説明する。

【0040】

本実施の形態においては、CPU 1001 によって、プログラム 1201 に記述される処理 1202、1203、1204 が、処理 1202、処理 1203、処理 1204 の順番に処理され、かつ、処理 1202、1203 はバス 1002 を使用する。

【0041】

図 4、図 5 に示すように、プログラム 1201 のコンパイルの際には、まず、処理 1202、1203、1204 はコンパイル手段 1301 によってコンパイルされる。また同時に、処理識別手段 1302 によってバス 1002 を使用する処理として処理 1202、1203 が識別される。次に、処理識別手段 1302 によって得られた処理 1202 と処理 1203 の実行の直前に使用状況判定処理および置換処理が行われるように、処理付加手段 1303 によって使用状況判定処理および置換処理が付加される。前記付加処理により、プログラム 1201 の処理は、処理 1202 が行われる直前において、使用状況判定処理および置換処理が実行されることとなる。

【0042】

バスアクセス状態レジスタ 1102 に記録される値は、バスアクセス識別装置 1101 によって算出される。この算出される値は、図 6 に示すように、バスアクセスが発生していれば増加し、バスアクセスが発生していなければ減少し、値の増減の度合いは任意に設定することができる。

【0043】

ここで、算出される値を α とし、 α の増減の度合いを β とする。 α の算出方法は、まず、 α に 1 サイクル毎に β を掛けて、次に、その値 ($\alpha \times \beta$) に、バスア

アクセスが発生していれば1を、バスアクセスが発生していなければ0を足す。つまり、nサイクル目での α の値を $\alpha(n)$ とすると、

$$\alpha(n) = \beta \times \alpha(n-1) + (1 \text{ または } 0)$$

となる。例えば、 β を0.9とした場合、 α の値は0から10の間の値をとることになり、その α の値がバスアクセスが頻繁に発生しているかどうかの目安の値となる。

【0044】

β を0.9とした場合のバスアクセスの状態を表す α とサイクル数の関係は図6のように示される。 α の値の変化が増加している箇所はバスサイクルが頻繁に発生しており、 α の値が減少している箇所はバスアクセスが発生していない。

【0045】

以上のように、バスアクセス識別装置1101は、バスアクセスが頻繁に発生しているかどうかの目安となる α の値を算出し、その結果をバスアクセス状態レジスタ1102へ記録する。

【0046】

使用状況判定処理は、バスアクセス状態レジスタ1102に記録されている値 α を読み取り、特定の値と比較してバスアクセスの状態を判定する。この特定の値は任意に設定することができる値であり、バスアクセスが頻繁に発生しているかどうかを判定する閾値となる。前述の $\beta = 0.9$ の場合、 α は0から10の値をとることになるので、前述の閾値を5と設定したとすると、 α が5より大きければ、その時点でバスアクセスが頻繁に発生していると判定できる。逆に α が5以下であれば、バスアクセスは頻繁に発生していないと判定できる。

【0047】

この判定の結果を受けて、置換処理は、処理1202を処理1205へ変更する。つまり、判定の結果、バス1002が頻繁に使用されていれば、置換処理によって処理1202の代わりに処理1205が行われる。逆に、判定の結果、バス1002が頻繁に使用されていなければ、処理1202がそのまま行われる。

【0048】

処理識別手段1302によってバス1002を使用する処理であると識別され

た処理 1203 についても同様に、処理付加手段 1303 によって付加された使用状況判定処理および置換処理が行われ、バス 1002 が頻繁に使用されていれば、処理 1203 の代わりに処理 1206 が行われ、バス 1002 が頻繁に使用されていなければ、処理 1203 が行われる。

【0049】

以上のように本実施の形態によれば、使用状況判定処理および置換処理によって、バス 1002 におけるバスアクセス競合の発生を軽減し、バス 1002 を効率良く使用することが可能となり、バスアクセスの競合の発生によるシステムの処理速度の低下の影響を軽減することができる。

【0050】

なお、本実施の形態では、プロセッサを 1 つの CPU、リソースを 1 つのバスとしたが、それぞれ複数の CPU、バスでも良く、少なくとも 1 つのリソースとそのリソースを使用する少なくとも 1 つのプロセッサであれば良いものである。

【0051】

なお、本実施の形態では、バスアクセス状態レジスタに α を求めているが、 α の求め方あるいはバスのアクセス頻度を求める方法は本方法によらずともよく、指標として求めることができるものであれば同等の効果を得られる。

【0052】

(第 2 の実施の形態)

第 2 の実施の形態によるソフトウェア処理システムを図 8 ～ 図 15 を用いて具体的に説明する。

【0053】

図 8 は、第 2 の実施の形態にかかわるソフトウェア処理システムのハードウェアの構成を示すブロック図である。図 8 においてソフトウェア処理システムは、プロセッサ 2001 と、DSP 2002 と、プロセッサ 2001 と DSP 2002 で実行するソフトウェアを格納した記憶装置 2003 と、周辺回路 2004 と、バス調停回路 2005 と、外部メモリ 2006 と、共有バス 2007 で構成される。

【0054】

図8では、プロセッサ2001はDSP2002を突き放し制御方式で制御しており、プロセッサ2001が記憶装置2003から読み出したソフトウェアがDSP2002で実行すべき処理の場合は、プロセッサ2001はDSP2002へ処理要求を発行し、DSP2002は処理要求を受けた後、処理を実行する。外部メモリ2006はプロセッサ2001、DSP2002が処理を実行する際に使用する共有メモリである。バス調停回路2005はバスアクセスを管理しており、共有メモリへのアクセス競合、すなわち共有バス2007のバス競合が発生した場合には、外部メモリ2006へアクセスする処理を実行するか否かを判定する。

【0055】

ここで、バス調停回路2005の制御フローを図9に示す。図9において、2101は共有メモリである外部メモリ2006へのアクセス要求（読み出しまたは書き込み要求）が発生したか否かを判定するアクセス要求判定手段であり、2102は現在、外部メモリ2006がアクセス中か否かを判定するメモリ状態判定手段であり、2103はアクセス要求に対しバスを解放しメモリアクセスを促すメモリアクセス許可手段であり、2104は処理が終了したか否かを判定する処理終了判定手段である。

【0056】

外部メモリ2006へのアクセス中に新たな外部メモリ2006へのアクセス要求が発生した場合、つまりバス競合が発生した場合、バス調停回路2005は、一旦、アクセス要求を受理し、実際の処理に関しては先の処理が終了するまで待機（wait）状態となる。外部メモリ2006へのアクセスが頻繁に発生すると、バス競合が発生し、バス調停回路2005により処理が待機（wait）状態となる。その結果、ソフトウェア処理システム全体の処理速度が低下する。そこで第2の実施の形態では、バス競合を避け、処理速度の低下を防ぐ。

【0057】

本実施の形態におけるコンパイラは、処理記述言語で記述されたソフトウェアからオブジェクトコードを出力する通常のコンパイラとしての機能の他に、図10の処理フローに示す機能を有する。コンパイラは、まず、処理識別手段220

1として外部メモリ2006へ頻繁にアクセスする処理を識別する。これは、処理識別手段2201が外部メモリ2006へアクセスする処理をあらかじめ認識していれば可能である。次に、識別した処理に対して、これと等価であるが、外部メモリ2006へのアクセスが少ない処理である等価処理があらかじめライブラリ等で用意されているか否かを判定する。もし用意されていれば、コンパイラは処理識別手段2201で識別した処理に対して等価処理、記憶処理、使用状況判定処理、置換処理を追加する。ここで、記憶処理とは外部メモリ2006へ頻繁にアクセスする処理の要求が発行されてから処理が終了するまでの処理時間を記録する処理であり、使用状況判定処理とは記憶処理で記憶した使用状況の情報からバス競合が発生しているか否かを判定する処理であり、置換処理とは処理識別手段2201で識別した処理を等価処理へ置き換える処理を示す。

【0058】

図11は、図10の処理フローをもつコンパイラを使用したときの外部メモリ2006へ頻繁にアクセスする処理の変更を示す図である。図11では、外部メモリ2006へ頻繁にアクセスする処理を関数“mem_acc_many ()”とし、この等価処理を“mem_acc_few ()”としている。コンパイラは、あらかじめ外部メモリ2006へアクセスする処理を登録することで、処理識別手段2201を用いてソフトウェアの中から関数“mem_acc_many ()”を識別する。

【0059】

次に、コンパイラは処理識別手段2201で識別した処理に対して上記の等価処理、記憶処理、使用状況判定処理、置換処理を付加する。

【0060】

まず、図11(b)の2行目では、ソフトウェアに使用状況判定処理を付加する。ここで“task_time”はプロセッサ2001が外部メモリ2006へ頻繁にアクセスする処理の要求を発行してから処理が終了するまでの処理時間であり、“DEFINED_TIME”はあらかじめ設定された設定値である。処理時間“task_time”は、過去に発生した外部メモリ2006へ頻繁にアクセスする処理の処理時間を表しており、現時刻に発生する外部メモリ20

06へ頻繁にアクセスする処理を実行する前に“task_time”を参照し、設定値である“DEFINED_TIME”と比較することでバス競合を判定する。バス競合が発生中と判定された場合、図9の制御フローに従うバス調停回路2005では、バス競合の発生時に処理を待機状態とするため、そのときの処理時間は長くなる。つまり、外部メモリ2006へ頻繁にアクセスする処理の処理時間がある程度大きくなるとバス競合が発生したと推定できる。なお、設定値については、ソフトウェア処理システムの仕様に応じて的確な値を設定するのが望ましい。

【0061】

次に、図11(b)の3行目から6行目は、記憶処理を示す。使用状況判定処理によりバス競合が発生していないと推定された場合、“mem_acc_many ()”実行の前後のタイマーの値を参照し、その差を算出することで処理時間“task_time”を算出している。

【0062】

図11(b)の8行目から10行目は、置換処理を示す。使用状況判定処理によりバス競合が発生していると推定された場合、さらにバス競合を発生させることを防ぐため、外部メモリ2006へのアクセスが少ない処理に置き換える。なお、置換処理で置換する処理は、処理の整合性を保つために機能的に等価であるが、外部メモリ2006へのアクセスが少ない処理が望ましい。

【0063】

第2の実施の形態では、上述のコンパイラが追加した等価処理、記憶処理、使用状況判定処理、置換処理により、使用状況監視処理の過程およびソフトウェア処理変更処理の過程を実現する。つまり、使用状況監視処理の過程とは、記憶処理により処理時間“task_time”を記憶しており、使用状況判定処理へ処理時間“task_time”を出力する。ソフトウェア処理変更処理の過程とは、識別した処理を等価処理へ置き換える置換処理であり、使用状況監視処理の過程から出力される処理時間に基づいてバス競合を判定し、バス競合が発生していれば処理の置き換えを行う。

【0064】

ここで、本実施の形態によるバス競合の判定フローを図12に示す。処理識別手段2201で識別した処理に対し、前回識別した処理を実行した際の処理時間“task_time”と設定値“DEFINED_TIME”を比較する。処理時間“task_time”が設定値“DEFINED_TIME”以上の場合には、使用状況判定処理はバス競合が発生中であると判定し、等価処理への置き換えを行う。逆に処理時間“task_time”が設定値“DEFINED_TIME”よりも小さい場合には、使用状況判定処理はバス競合が発生していないと判定し、処理識別手段2201で識別した処理を、処理時間“task_time”の記憶を行いながら実行する。

【0065】

ここで、「前回識別した処理を実行した際の処理時間」とは、2つ考えられる。1番目の処理時間は、処理識別手段2201で識別した処理の中で、現時刻から1回前に記録した処理時間、つまり、記録した処理時間の中で最も新しいものが該当する。2番目の処理時間は、現時刻で識別した処理と同一の処理を過去に実行したときに記録した処理時間が該当する。1番目の処理時間では、処理時間は常に最新のものを1つだけ記憶するが、2番目の処理時間では、識別された処理毎に処理時間を記憶する必要がある。

【0066】

以上、本実施の形態では、処理時間“task_time”と設定値“DEFINED_TIME”の比較結果より、バス競合が発生中か否かを判定し、バス競合が発生中と判定された場合は、外部メモリ2006へ頻繁にアクセスする処理を外部メモリ2006へのアクセスが少ない処理に自動で置換する。その結果、動的にバス競合を判定しながら自動で処理の割り当てを行うことができるとともに、新たなバス競合を避けることができ、処理速度の低下を防止できる。なお、第2の実施の形態における上記の発明内容を後述の発明と区別するために第1の手法とする。

【0067】

さらに、第2の手法としての使用状況判定処理においては、バス調停回路2005から出力された競合情報である処理時間“task_time”と事前に設

定しておいた設定値“DEFINED_TIME”との比較結果に加えて、乱数を使用することでバス競合の判定を行う。

【0068】

図13は、処理時間“task_time”と設定値“DEFINED_TIME”の比に対する等価処理への置き換えを行う確率を示している。処理時間“task_time”が設定値“DEFINED_TIME”よりも小さい場合、言い換えれば、図13で処理時間“task_time”と設定値“DEFINED_TIME”の比が1より小さい場合、使用状況判定処理は第1の手法と同様にバス競合が発生していないと判定し、等価処理の置き換えを行わない。

【0069】

しかし、処理時間“task_time”が設定値“DEFINED_TIME”以上の場合、言い換えれば、図13で処理時間“task_time”と設定値“DEFINED_TIME”の比が1以上の場合、図13の確率に従って等価処理への置き換えを行ったり、あるいは等価処理への置き換えは行わずに、処理時間を再計測し記憶する。これは、処理時間“task_time”が設定値“DEFINED_TIME”以上の場合、第1の手法では必ずバス競合が発生中と判定していたが、第2の手法では乱数を用いた確率に応じてバス競合を判定するので、第1の手法とは逆の判定結果を出力する可能性を持たせた。逆の判定結果とは、バス競合が発生していないという判定結果である。

【0070】

そして、本実施の形態では図12に示すバス競合判定フローに従い、処理時間を再度記録することになる。つまり、バス競合の判定に乱数を加えることで判定を見直しする。これにより、実際にはバス競合が解消されたにもかかわらず、使用状況判定処理はバス競合がそのまま継続されている、といった間違った判断を避けることができる。

【0071】

第2の手法は一連の処理の中でバス競合が頻繁に発生するシステムに対して有効である。なお、図13では、処理時間“task_time”と設定値“DEFINED_TIME”の比が大きければ大きいほど、つまり、処理時間“ta

“s k _ t i m e” が設定値よりも大きな値をとればとるほど等価処理へ置き換える確率が小さくなっている。これは、処理時間 “t a s k _ t i m e” が長ければ長いほど近い未来にバス競合は解消されるということが一般的に考えられるからである。

【0072】

なお、図13の発生確率の値については、対象システムの仕様に応じて適当な値を与える必要がある。

【0073】

以上、第2の手法により使用状況判定処理におけるバス競合の判定精度を向上できる。

【0074】

なお、本実施の形態の第2の手法では、バス競合の判定について乱数を用いた確率を使用したか、乱数を用いないバス競合の判定も考えられる。例えば、バス競合時における処理時間の平均値または最大値などの定数があらかじめ既知であれば、バス競合中と判定されてから一定周期ごとに処理時間の再計測及び記録を行うことも可能である。

【0075】

次に、第3の手法について説明する。

【0076】

外部メモリ2006へ頻繁にアクセスする処理が複数の箇所が存在する場合、その処理の呼び出し元（出現箇所）別にバス競合を判定すれば判定精度は向上する。第3の方法では、処理識別手段2201で識別した処理の出現箇所毎に処理時間を記憶し、それを使用することでバス競合の判定を高精度に行う。

【0077】

図8のソフトウェア処理システムでは、複数の処理が記述された関数を実行する際には、現在のプログラムカウンタを退避レジスタに一時的に退避させた後、関数の処理を実行し、関数の処理が終了した後、退避レジスタからプログラムカウンタを読み出してプログラムカウンタを元の値に戻す操作を行う。第3の手法では、出現箇所識別処理が上述の退避レジスタからプログラムカウンタを読み出

すことで処理識別手段 2201 で識別した処理の出現箇所を特定する。

【0078】

図 12 の判定フローをもつ第 1 の手法に、さらに出現箇所識別処理を追加した場合のバス競合の判定フローを図 14 に示す。

【0079】

まず、処理識別手段 2201 で識別された処理に対し、出現箇所識別処理により処理の出現箇所を調査する。上述したように退避レジスタを参照することで、出現箇所の特定は可能である。次に、出現箇所毎に記憶した処理時間を参照する。次に、出現箇所毎に記憶した処理時間を設定値と比較することでバス競合が発生中か否かを判定し、バス競合が発生中であれば等価処理に置き換えて処理を実行し、バス競合が発生していなければ、処理時間を記憶しながら識別した処理をそのまま実行する。なお、出現箇所毎に処理時間を記憶する部分に処理時間が記憶されていない場合は、バス競合が発生していないと判定されたとみなし、処理時間を計測しながら処理識別手段 2201 で識別された処理を実行し、計測した処理時間を新規に記憶する。

【0080】

第 3 の手法は、ループ処理など同一の処理が繰り返して処理される場合、出現箇所別に判定した処理時間を参照することでバス競合の判定精度が向上する。

【0081】

なお、本実施の形態における処理時間 “task_time” を、第 1 の実施の形態の使用状況監視装置に記憶することでバス競合の判定を行うことも可能である。このときのバス競合の判定フローを図 15 に示す。処理時間を求めた後、第 1 の実施の形態における使用状況監視装置にも処理時間を記憶することにより、バス競合を動的に判定でき、バス競合に伴う処理速度の低下を防ぐことが可能である。

【0082】

(第 3 の実施の形態)

図 16 は第 3 の実施の形態にかかわるソフトウェア処理システムのハードウェアの構成を示すブロック図である。図 16 において、ソフトウェア処理システム

は、DSP 4001とプロセッサ4002と、DSP使用状況監視装置4003と、DSP 4001とプロセッサ4002で実行する処理である図17のオブジェクトコード4105を格納したメモリ4004で構成される。メモリ4004は、DSP 4001とプロセッサ4002が処理を実行する際の共有メモリとして処理結果や処理データを記憶する。

【0083】

図17は、本発明の第3の実施の形態にかかわるコンパイラの処理フローを示す。

【0084】

図18は、本発明の第3の実施の形態にかかわるコンパイラによる処理の変更を示す。

【0085】

図19は、本発明の第3の実施の形態にかかわるDSP競合の判定フローを示す。

【0086】

図20は、本発明の第3の実施の形態にかかわるソフトウェア処理システムにおける処理の流れを示す。

【0087】

第3の実施の形態によるソフトウェア処理システムを図16～図20を用いて具体的に説明する。

【0088】

図16では、プロセッサ4002はDSP 4001を突き放し制御方式で制御しており、プロセッサ4002がメモリ4004から読み出したソフトウェアがDSP 4001で実行する処理の場合は、プロセッサ4002はDSP 4001へ処理要求を発行し、DSP 4001は処理要求を受けた後、処理を実行する。ここで、DSP使用状況監視装置4003は、現在、DSP 4001で処理が実行中か否かを記憶したレジスタ（記憶装置）であり、特定のアドレスが割り当てられているため、ソフトウェアの処理から読み出し可能である。

【0089】

図17はソフトウェアのコンパイラのフローである。C言語などの処理記述言語で書かれたソースコード4102は、プロセッサ用ライブラリ4103とDSP用ライブラリ4104を用いて、オブジェクトコード4105に変換される。その結果、オブジェクトコード4105には実行に必要なプロセッサ用ライブラリ4106とDSP用ライブラリ4107が含まれる。

【0090】

本実施の形態のコンパイラは、処理記述言語からオブジェクトコードを生成する通常のコンパイラの機能の他に、ソースコード4102からDSP4001で実行する処理を識別する処理識別手段と、処理識別手段で識別した処理に対し特定の処理を付加する処理付加手段を備える。処理識別手段および処理付加手段による処理の変更を図18に示す（左の数字は行番号）。ここで、図18（a）の処理記述言語で書かれたソースコードは、処理識別手段を用いて、ソースコード4102からDSP4001で実行されるとして識別された処理を表している。つまり、関数“func1（）”、“func2（）”、“func3（）”は全てDSP4001で実行される処理である。なお、DSP4001で実行する処理をあらかじめ処理識別手段に登録することでソースコードからDSP4001で実行する処理を識別可能である。

【0091】

図18（b）の処理記述言語で書かれたソースコードは、処理付加手段で付加された処理を示す。なお、便宜上、図18（b）では、処理付加手段はfunc2のみに特定の処理を付加しているが、処理付加手段は、処理識別手段で識別した全ての処理に対して特定の処理を付加しており、func1、func3についてもfunc2と同様に特定の処理が付加される。

【0092】

処理付加手段により付加される特定の処理とは、使用状況判定処理と置換処理である。使用状況判定処理とは、DSP4001が処理を実行中か否かをDSP使用状況監視装置4003を読み出すことで判定する処理である。図18（b）では、8行目の“if（DSPが待ち状態）”が使用状況判定処理に相当する。ここで待ち状態とは、DSP4001は処理を実行しておらず、プロセッサ40

02からの処理要求を待っている状態を表す。図18(b)では9行目から11行目が相当し、func2の処理内容として、DSP用ライブラリ4104を使用した場合を“func2_dsp()”、プロセッサ用ライブラリ4103を使用した場合を“func2_cpu()”としている。図18では、置換後の処理としてプロセッサで同等の機能をもつ処理を行った場合を示す。

【0093】

つまり、本実施の形態では、DSP4001を使用する処理を識別し、その処理を実行する前にDSP使用状況監視装置4003を観測することでDSP4001の状態を判定する。そして、判定結果を受けて、DSP4001が待ち状態であれば、そのままDSP4001で処理を実行するが、DSP4001が他の処理を実行中の場合は、DSP4001を使用しない他の処理に置換する。このようなソフトウェア処理変更処理により、DSP競合を避けることができる。参考として、図19に本実施の形態のDSP競合の判定フローを示す。

【0094】

次に具体例として、本実施の形態のソフトウェア処理システムにおける処理の流れを図20に示す。

【0095】

図20(a)は、DSP競合が発生しない場合における本実施の形態のソフトウェア処理システムの処理の流れである。ソフトウェアには、func0, func1, func2, func3, func4, func5, func6の処理があり、func0, func2, func4については、プロセッサ4002からDSP4001へ処理要求を発行し、それぞれDSP用ライブラリ4104を用いてfunc0_dsp, func2_dsp, func4_dspをDSP4001上で実行する。また、DSP4001とプロセッサ4002は共有メモリを用いて互いの処理結果を受け渡しており、func0_dspとfunc2_dspとfunc4_dspとfunc5は処理結果を共有メモリ4004に格納し、格納された処理結果は、func3とfunc5が処理を実行する際に読み出す。図20(a)のようにDSP競合が発生しない場合には、プロセッサ4002とDSP4001の資源を無駄なく使用しながら処理を実行する。

【0096】

しかし、図20(b)のようにfunc0_dspの処理時間が長くなった場合、func1がfunc2_dspの処理要求を発行した場合、DSP競合が発生し、func0_dspが終了するまでfunc2_dspは処理を開始できない。その結果、func2_dspの処理結果を用いるfunc3も実行できなくなり、処理全体の速度が遅くなってしまう。

【0097】

そこで、本実施の形態のソフトウェア処理システムでは、func1がfunc2_dspの処理要求を発行した際に、DSP使用状況監視装置4003を観測することでDSP競合を認識し、func2と等価な機能でありDSP4001でなくプロセッサ4002上で処理を行うfunc2_cpuをfunc2_dspの代わりに実行する。つまり、図20(b)では、DSP4001で実行する処理の処理要求を発行する際にDSP4001の状態を監視し、処理をDSP4001かプロセッサ4002に的確に割り振るソフトウェア処理変更処理を用いる。これによりDSP競合を防ぐことができ、処理速度の低下を防ぐ。さらに、処理を的確に割り当てて待ち状態のプロセッサ4002を使用することで、資源を効率的に使用可能である。

【0098】

なお、本実施の形態のコンパイラの処理識別手段、処理付加手段により、処理サイクル数は増えるが、DSP使用状況監視装置4003以外、新規にハードウェアを追加することなく、DSP競合を避け、処理速度の低下を防止できる。

【0099】

(第4の実施の形態)

第4の実施の形態によるソフトウェア処理システムを図21～図28を用いて具体的に説明する。

【0100】

図21は、第4の実施の形態にかかわるソフトウェア処理システムのハードウェアの構成を示すブロック図である。図21においてソフトウェア処理システムは、プロセッサ5001と、DSP5002と、プロセッサ5001とDSP5

002で実行するソフトウェアを格納した記憶装置5003と、周辺回路5004と、バス調停回路5007と、外部メモリ5005と、共有バス5006と、調停回路5008と、使用状況監視装置5009で構成される。図21では、プロセッサ5001はDSP5002を突き放し制御方式で制御しており、プロセッサ5001が記憶装置5003から読み出したソフトウェアがDSP5002で実行すべき処理の場合はプロセッサ5001はDSP5002へ処理要求を発行し、DSP5002は処理要求を受けた後、処理を実行する。外部メモリ5005はプロセッサ5001、DSP5002が処理を実行する際に使用する共有メモリである。調停回路5008は、DSP5002で処理を実行中に新たな処理要求が発行された場合、すなわちDSP競合が発生した場合に、新たに発行された処理が実行できるか否かを判定する。また、使用状況監視装置5009はDSP5002の使用状況を監視し、要求に応じて使用状況を出力する。

【0101】

ここで、調停回路5008の制御フローを図22に示す。図22において、5101はDSP5002で実行する処理の処理要求が発生したか否かを判定する処理要求判定手段であり、5102は現在、DSP5002がプロセッサ5001からの処理要求を待っている待ち状態か否かを判定するDSP状態判定手段であり、5103は処理要求に対しDSP5002での処理の実行を促すDSP使用許可手段であり、5104は処理が終了したか否かを判定する処理終了判定手段である。DSP5002で処理を実行中に新たなDSP5002への処理要求が発生した場合、つまり、DSP競合が発生した場合、調停回路5008は、一旦、処理要求を受理し、実際の処理に関しては先の処理が終了するまで待機（wait）状態となる。DSP5002で実行する処理が頻繁に発生すると、DSP競合が発生し、調停回路5008により処理が待機（wait）状態となる。その結果、ソフトウェア処理システム全体の処理速度が低下する。そこで第4の実施の形態では、DSP競合を避け、処理速度の低下を防ぐ。

【0102】

図23は、本発明の第4の実施の形態におけるソフトウェアのコンパイラの処理フロー図を示す。コンパイラは、まず、処理識別手段5201としてDSP5

002で実行する処理を識別する。これは、処理識別手段5201がDSP5002で実行する処理をあらかじめ認識していれば可能である。次に、識別した処理に対して、これと機能的には等価であるが、プロセッサ5001で実行する処理である等価処理があらかじめライブラリ等で用意されているか否かを判定する。もし用意されていれば、コンパイラは処理識別手段5201で識別した処理に対して等価処理、記憶処理、使用状況判定処理、置換処理を追加する。ここで記憶処理とはDSP5002で実行処理の処理要求が発行されてから処理が開始するまでの待ち時間を記録する処理であり、使用状況判定処理とは記憶処理で記憶した使用状況の情報からDSP競合が発生しているか否かを判定する処理であり、置換処理とは処理識別手段5201で識別した処理を等価処理へ置き換える処理を示す。

【0103】

図24は、図23の処理フローをもつコンパイラを使用したときのDSP5002で実行する処理に対する変更を示す図である。図24では、DSP5002で実行する処理を関数“dsp_task ()”とし、この等価処理を“proc_dsp_task ()”としている。コンパイラは、あらかじめDSP5002で実行する処理を登録することで処理識別手段5201を用いてソフトウェアの中から関数“dsp_task ()”を識別する。次に、コンパイラは処理識別手段5201で識別した処理に対して上記の等価処理、記憶処理、使用状況判定処理、置換処理を付加する。

【0104】

まず、図24 (b) の2行目では、ソフトウェアに使用状況判定処理を付加する。ここで“wait_time”はプロセッサ5001がDSP5002で実行する処理の処理要求を発行してから処理が開始するまでの待ち時間であり、“DEFINED_TIME”はあらかじめ設定された設定値である。待ち時間“wait_time”は、過去に発生したDSP5002で実行する処理の待ち時間を表しており、現時刻に発生するDSP5002で実行する処理要求を発行する前に“wait_time”を参照し、設定値である“DEFINED_TIME”と比較することでDSP競合を判定する。DSP競合が発生中と判定され

た場合、図 22 の制御フローに従う調停回路 5008 では、DSP 競合の発生時に処理を待機状態とするため、そのときの待ち時間は長くなる。つまり、DSP 5002 で実行する処理の待ち時間がある程度大きくなると DSP 競合が発生したと推定できる。なお、設定値については、ソフトウェア処理システムの仕様に応じて的確な値を設定するのが望ましい。

【0105】

次に、図 24 (b) の 3 行目から 6 行目は、記憶処理を示す。使用状況判定処理により DSP 競合が発生していないと推定された場合、処理要求を発行した時点と “dsp_task ()” の実行が開始される時点のタイマーの値を参照し、その差を算出することで待ち時間 “wait_time” を算出している。

【0106】

図 24 (b) の 9 行目から 11 行目は、置換処理を示す。使用状況判定処理により DSP 競合が発生していると推定された場合、さらに、DSP 競合を発生させることを防ぐため、プロセッサ 5001 で実行する等価処理に置き換える。なお、置換処理で置換する処理は、処理の整合性を保つために機能的に等価であるが、DSP 5002 で実行しない処理が望ましい。

【0107】

第 4 の実施の形態では、上述のコンパイラが追加した等価処理、記憶処理、使用状況判定処理、置換処理により、使用状況監視処理の過程およびソフトウェア処理変更処理の過程を実現する。つまり、使用状況監視処理の過程とは、記憶処理により待ち時間 “wait_time” を記憶しており、使用状況判定処理へ待ち時間 “wait_time” を出力する。ソフトウェア処理変更処理の過程とは、識別した処理を等価処理へ置き換える置換処理であり、使用状況監視装置から出力される待ち時間に基づいて DSP 競合を判定し、DSP 競合が発生していれば処理の置き換えを行う。

【0108】

ここで、本実施の形態による DSP 競合の判定フローを図 25 に示す。処理識別手段 5201 で識別した処理に対し、前回識別した処理を実行した際の待ち時間 “wait_time” と設定値 “DEFINED_TIME” を比較する。

待ち時間 “wait_time” が設定値 “DEFINED_TIME” 以上の場合には、使用状況判定処理はDSP競合が発生中であると判定し、等価処理への置き換えを行う。逆に待ち時間 “wait_time” が設定値 “DEFINED_TIME” よりも小さい場合には、使用状況判定処理はDSP競合が発生していないと判定し、処理識別手段5201で識別した処理を待ち時間 “wait_time” を記憶しながら実行する。

【0109】

ここで「前回識別した処理を実行した際の待ち時間」とは、2つ考えられる。1番目の待ち時間は処理識別手段2201で識別した処理の中で、現時刻から1回前に記録した待ち時間、つまり、記録した待ち時間の中で最も新しいものが該当する。2番目の待ち時間は、現時刻で識別した処理と同一の処理を過去に実行したときに記録した待ち時間が該当する。1番目の待ち時間では、待ち時間は常に最新のものを1つだけ記憶するが、2番目の待ち時間では、識別された処理毎に待ち時間を記憶する必要がある。

【0110】

以上、本実施の形態では、待ち時間 “wait_time” と設定値 “DEFINED_TIME” の比較結果より、DSP競合が発生中か否かを判定し、DSP競合が発生中と判定された場合は、DSP5002で実行する処理をプロセッサ5001で実行する処理に自動で置換する。その結果、動的にDSP競合を判定しながら自動で処理の割り当てを行うことができるとともに、新たなDSP競合を避けることができ、処理速度の低下を防止できる。なお、第4の実施の形態における上記の発明内容を後述の発明と区別するために第4の手法とする。

【0111】

さらに、第5の手法としての使用状況判定処理においては、使用状況監視装置から出力される競合情報である待ち時間 “wait_time” と事前に設定しておいた設定値 “DEFINED_TIME” との比較結果に加えて、乱数を使用することでDSP競合の判定を行う。

【0112】

図26は、待ち時間 “wait_time” と設定値 “DEFINED_TIME”

“E” の比に対する等価処理への置き換えを行う確率を示している。待ち時間 “wait_time” が設定値 “DEFINED_TIME” よりも小さい場合、言い換えれば、図 26 で待ち時間 “wait_time” と設定値 “DEFINED_TIME” の比が 1 より小さい場合、使用状況判定処理は第 4 の手法と同様に DSP 競合が発生していないと判定し、等価処理の置き換えを行わない。

【0113】

しかし、待ち時間 “wait_time” が設定値 “DEFINED_TIME” 以上の場合、言い換えれば、図 26 で待ち時間 “wait_time” と設定値 “DEFINED_TIME” の比が 1 以上の場合、図 26 の確率に従って等価処理への置き換えを行ったり、あるいは等価処理への置き換えは行わずに待ち時間を再計測し記憶する。これは、処理時間 “wait_time” が設定値 “DEFINED_TIME” 以上の場合、第 4 の手法では必ず DSP 競合が発生中と判定していたが、第 5 の手法では乱数を用いた確率に応じて DSP 競合を判定するので、第 4 の手法とは逆の判定結果を出力する可能性を持たせた。逆の判定結果とは、DSP 競合が発生していないという判定結果である。

【0114】

そして、本実施の形態では図 25 に示す DSP 競合判定フローに従い、待ち時間を再度記録することになる。つまり、DSP 競合の判定に乱数を加えることで判定を見直しする。これにより、実際には DSP 競合が解消されたにもかかわらず、使用状況判定処理は DSP 競合がそのまま継続されている、といった間違っただ判断を避けることができる。

【0115】

第 5 の手法は一連の処理の中で DSP 競合が頻繁に発生するシステムに対して有効である。なお、図 26 では、待ち時間 “wait_time” と設定値 “DEFINED_TIME” の比が大きければ大きいほど、つまり、待ち時間 “wait_time” が設定値よりも大きな値をとればとるほど等価処理へ置き換える確率が小さくなっている。これは、待ち時間 “wait_time” が長ければ長いほど近い未来に DSP 競合は解消されるということが一般的に考えられるからである。

【0116】

なお、図26の発生確率の値については、対象システムの仕様に応じて適当な値を与える必要がある。

【0117】

以上、第5の手法により使用状況判定処理におけるDSP競合の判定精度を向上できる。

【0118】

なお、本実施の形態の第5の手法では、DSP競合の判定について乱数を用いた確率を使用したのが、乱数を用いないDSP競合の判定も考えられる。例えば、DSP競合時における待ち時間の平均値または最大値などの定数があらかじめ既知であれば、DSP競合中と判定されてから一定周期ごとに待ち時間の再計測及び記録を行うことも可能である。

【0119】

次に、第6の手法について説明する。

【0120】

DSP5002で実行する処理が複数の箇所が存在する場合、その処理の呼び出し元（出現箇所）別にDSP競合を判定すれば判定精度は向上する。第6の方法では、処理識別手段5201で識別した処理の出現箇所毎に待ち時間を記憶し、それを使用することでDSP競合の判定を高精度に行う。

【0121】

図21のソフトウェア処理システムでは、複数の処理が記述された関数を実行する際には、現在のプログラムカウンタを退避レジスタに一時的に退避させた後、関数の処理を実行し、関数の処理が終了した後、退避レジスタからプログラムカウンタを読み出してプログラムカウンタを元の値に戻す操作を行う。第6の手法では、出現箇所識別処理が上述の退避レジスタからプログラムカウンタを読み出すことで処理識別手段5201で識別した処理の出現箇所を特定する。

【0122】

図25の判定フローをもつ第4の手法に、さらに出現箇所識別処理を追加した場合のDSP競合の判定フローを図27に示す。

【0 1 2 3】

まず、処理識別手段 5 2 0 1 で識別された処理に対し、出現箇所識別処理により処理の出現箇所を調査する。上述したように退避レジスタを参照することで、出現箇所の特定は可能である。次に、出現箇所毎に記憶した待ち時間を参照する。次に、出現箇所毎に記憶した待ち時間を設定値と比較することで D S P 競合が発生中か否かを判定し、D S P 競合が発生中であれば等価処理に置き換えて処理を実行し、D S P 競合が発生していなければ待ち時間を記憶しながら識別した処理をそのまま実行する。なお、出現箇所毎に記憶した待ち時間に待ち時間が記憶されていない場合は、D S P 競合が発生していないと判定されたとみなし、待ち時間を計測しながら処理識別手段 5 2 0 1 で識別された処理を実行し、計測した待ち時間を新規に記憶する。

【0 1 2 4】

第 6 の手法は、ループ処理など同一の処理が繰り返して処理される場合、出現箇所別に判定した待ち時間を参照することで D S P 競合の判定精度が向上する。

【0 1 2 5】

なお、本実施の形態における待ち時間 “wait_time” を、第 3 の実施の形態の使用状況監視装置に記憶することで D S P 競合の判定を行うことも可能である。このときの D S P 競合の判定フローを図 2 8 に示す。待ち時間を求めた後、第 1 の実施の形態における使用状況監視装置にも待ち時間を記憶することにより、D S P 競合を動的に判定でき、D S P 競合に伴う処理速度の低下を防ぐことが可能である。

【0 1 2 6】

(第 5 の実施の形態)

図 2 9 は第 5 の実施の形態にかかわるソフトウェア処理システムのハードウェアの構成を示すブロック図である。図 2 9 において、ソフトウェア処理システムは、D S P 7 0 0 1 とプロセッサ 7 0 0 2 と、D S P 使用状況監視装置 7 0 0 3 と、D S P 7 0 0 1 とプロセッサ 7 0 0 2 で実行する処理である図 3 0 のオブジェクトコードを格納したメモリ 7 0 0 4 で構成される。メモリ 7 0 0 4 は、D S P 7 0 0 1 とプロセッサ 7 0 0 2 が処理を実行する際の共有メモリとして処理結

果や処理データを記憶する。

【0127】

図30は第5の実施の形態にかかわるソフトウェア処理システムで実行するソフトウェアを示す。

【0128】

図31は第5の実施の形態にかかわるDSP競合時の処理フローを示す。

【0129】

図32は第5の実施の形態にかかわるソフトウェア処理システムにおける処理の流れを示す。

【0130】

第5の実施の形態によるソフトウェア処理システムを図29～図32を用いて具体的に説明する。

【0131】

図29では、プロセッサ7002はDSP7001を突き放し制御方式で制御しており、プロセッサ7002はメモリ7004からソフトウェアをフェッチ、デコードし、DSP7001で実行する処理の場合、プロセッサ7002はDSP7001へ処理要求を発行することでDSP7001は処理を実行する。ここで、プロセッサ7002からDSP7001への処理要求の発行について、さらに詳しく説明する。

【0132】

プロセッサ7002は、まず、DSP7001にあるコマンドバッファに処理コマンドおよび処理を行うのに必要なデータを書き込む。データ書き込みの終了後、プロセッサ7002はDSP7001へ処理の開始要求を発行し、それを受けたDSP7001は、コマンドバッファから処理コマンド、データを読み出すことで処理が実行される。つまり、第5の実施の形態において、プロセッサ7002からDSP7001への処理要求とは、DSP7001のコマンドバッファへの処理コマンド、データの書き込みおよび処理の開始要求の発行という一連の動作を意味する。

【0133】

また、図 29 における DSP 7001 の使用状況監視装置 7003 は、DSP 7001 が現在処理を実行中か否かを監視するとともに、プロセッサ 7002 が DSP 7001 のコマンドバッファへ処理コマンドおよびデータの書き込みを行うか否かを監視する。そして、DSP 7001 が任意の処理を実行中にプロセッサ 7002 が処理要求を発行するために DSP 7001 へのコマンドバッファへの書き込みを行った場合、DSP 使用状況監視装置 7003 はプロセッサ 7002 へ割り込み信号を与える。DSP 使用状況監視装置 7003 からの割り込み信号を受けたプロセッサ 7002 は、割り込みルーチンに分岐し、プロセッサ 7002 から DSP 7001 への処理の開始要求の発行を止める。その結果、DSP 7001 のコマンドバッファへは処理コマンド、データが格納されているが、DSP 7001 への処理の開始要求が止められるため、DSP 7001 は新たに処理を実行しない。つまり、割り込み信号により新規の DSP 7001 の処理要求がキャンセルされる。

【0134】

さらに、割り込み信号を受けたプロセッサ 7002 は、DSP 7001 で実行する処理の代わりに代替処理を行う。ここで代替処理とは、DSP 7001 を使用しない処理であり、本実施の形態では DSP 7001 で実行する予定だった処理と機能的には等価で、プロセッサ 7002 上で実行する処理とする。

【0135】

以上の割り込み信号に対する動作を実現するために、本実施の形態のソフトウェア処理システムでは、DSP 7001 で実行する処理に対し、それと機能的に等価であるが、プロセッサ 7002 で実行する処理をそれぞれ用意する。そして、DSP 使用状況監視装置 7003 による割り込み信号を受けた場合、プロセッサ 7002 は DSP 7001 への処理要求を止め、等価な処理をプロセッサ 7002 自身で代わりに実行する。

【0136】

図 30 に本実施の形態のソフトウェア処理システムにおけるソフトウェアを示す（左の数字は行番号）。main 関数の中に func1, func2, func3, func4, func5, func6, func7, func8 があり、

その中でDSP7001で実行する処理は、func2, func4, func6である。ソフトウェアにはプロセッサ7002上で実行する処理であるプロセッサ用ライブラリと、DSP7001上で実行する処理であるDSP用ライブラリが含まれており、図30におけるfunc2, func4, func6については、それぞれfunc2_dsp, func4_dsp, func6_dspというDSP用ライブラリを用いてDSP7001上で処理を実行できるし、func2_cpu, func4_cpu, func6_cpuというCPU用ライブラリを用いてプロセッサ7002上でも実行可能とする。

【0137】

図31は、図30のソフトウェアを実行した際のDSP競合時の処理フローであり、先に説明したようにDSP7001が他の処理を実行中にDSP7001に対して新規に処理要求が発行されようとしている場合、プロセッサ7002へ割り込み信号を与え、DSP用ライブラリを使用する処理をCPU用ライブラリを使用する処理に置き換える。

【0138】

図32は、本実施の形態のソフトウェア処理システムにおいて図30のソフトウェアを実行したときの処理の流れを示している。

【0139】

図32(a)はDSP7001のタスクが競合しない場合を示す。プロセッサ7002でのfunc1を実行する。プロセッサ7002はfunc1を実行後、DSP7001に対しfunc2の処理要求を発行し、DSP7001ではfunc2_dspを実行する。一方、プロセッサ7002はfunc3を実行する。DSP7001はfunc2を実行後、処理結果をメモリ7004に書き込む。なお、メモリ7004はプロセッサ7002、DSP7001からアクセス可能な共有メモリである。プロセッサ7002でfunc3を実行後、DSP7001でfunc4の処理要求を発行し、DSP7001でfunc4_dspを実行し、プロセッサ自身はfunc5を実行する。DSP7001はfunc4を実行後、処理結果をメモリ7004に書き込む。プロセッサ7002はfunc5を実行後、DSP7001でfunc6の処理要求を発行し、DSP70

01でfunc6_dspを実行し、プロセッサ自身はfunc7を実行する。
DSP7001はfunc6を実行後、処理結果をメモリ7004に書き込む。
プロセッサ7002は、以上の処理結果をメモリ7004から読み取り、func8を処理する。以上、図32(a)ではDSP7001のタスクが競合しないためプロセッサ7002とDSP7001の資源を効率良く使用できる。

【0140】

図32(b)では、DSP7001のタスクが競合する場合を示す。ここではfunc2_dspが終了しないうちに、プロセッサ7002からfunc4の処理要求を発行され、DSP競合が起きている。その結果、func4の処理はfunc2_dspが終了するまで停止させられるのでシステムの処理速度が低下する。

【0141】

そこで本実施の形態では、DSPの使用状況監視装置7003がDSP競合を判定し、func4の処理要求を発行しようとするプロセッサ7002へ割り込み信号を発生させる。それを受けたプロセッサ7002は、自身が実行中のfunc5を中断し、func4をCPU用ライブラリであるfunc4_cpuを用いてCPU上で処理し、メモリ7004に処理結果を書き込む。その後、プロセッサ7002は中断していたfunc5を再開する。

【0142】

このように、本実施の形態のソフトウェア処理システムでは、処理を実行しながらDSP競合を判定し、処理をDSP7001かプロセッサ7002に的確に割り振ることによって、DSP競合を避け、処理速度の低下を防ぐ。さらに、DSPの使用状況監視装置7003というハードウェアで処理することによって、割り込みルーチンを用意することを除いてソースコード4102に対する記述の修正、追加がないため、アセンブラやオブジェクトコードで提供された場合にも対応できる。

【0143】

(第6の実施の形態)

以下、本発明の第6の実施の形態を図面に基づいて説明する。

【0144】

図33は、本発明の第6の実施の形態にかかわるソフトウェア処理システムのハードウェアの構成を示すブロック図である。

【0145】

図33において、8001はCPU、8002はDSP、8003はバス、8004は外部メモリ、8005は周辺回路、8006はCPU8001が持つメモリ、8007はメモリ8006中のバンク形式であるDSP8002用のメモリバンク、8008はメモリ8006中のバンク形式であるCPU8001用のメモリバンク、8009はDSP8002が処理を行っているかどうかを監視する使用状況監視装置、8010はDSP用メモリバンク8007とCPU用メモリバンク8008とを切り替えるバンク切り替え装置である。

【0146】

図34は、本発明の第6の実施の形態にかかわるソフトウェア処理システムのソフトウェアの構成を示す図である。

【0147】

図34において、8101はCPU8001およびDSP8002上で動作するプログラム、8102、8103、8104はプログラム8101中に記述され、DSP8002を使用するコマンドを含んでライブラリ化された処理、8105、8106、8107は処理8102、8103、8104とそれぞれ同等の機能を持つ、CPU8001を使用するコマンドを含んでライブラリ化された処理である。

【0148】

図35は、本発明の第6の実施の形態にかかわるソフトウェア処理システムにおけるコンパイルフローを示すフローチャートである。

【0149】

図35において、8201はプログラム8101をコンパイルするコンパイル手段、8202はDSP8002で行われる処理を識別する処理識別手段、8203はDSP用メモリバンク8007またはCPU用メモリバンク8008にマッピングするマッピング手段である。

【0150】

以上のように構成した本実施の形態の動作について説明する。

【0151】

なお、図36は、本発明の第6の実施の形態にかかわるソフトウェア処理システムにおけるメモリバンクへのソフトウェアのマッピングを示す図である。

【0152】

本実施の形態において、DSP8002はCPU8001からの処理要求を受け付けることにより処理を実行し、CPU8001とDSP8002との間の制御方式は突き放し制御方式である。プログラム8101に記述される処理func1, func2, func3は、DSP8002またはCPU8001を使用するコマンドを含み、手順として処理func1、処理func2、処理func3の順番であって、それぞれの処理にはデータ依存がないものとする。

【0153】

プログラム8101が、コンパイラが有するコンパイル手段8201によってコンパイルされる際、処理識別手段8202によってDSP8002を使用するコマンドを含んだ処理である処理func1, func2, func3を識別し、図36に示すように、メモリバンクのマッピング手段8203によって、DSP用メモリバンク8007にはDSP8002用のライブラリ化された処理8102, 8103, 8104がマッピングされる。同様に、CPU用メモリバンク8008にはCPU8001用のライブラリ化された処理8105, 8106, 8107がマッピングされる。このとき、処理8102と処理8105は同じアドレス番地を開始アドレスとする。処理8103と処理8106、処理8104と処理8107についても同様である。このようにコンパイラは同じ名前である別々のプロセッサ用の処理を認識してコンパイルすることが可能であるという特徴を持つ。

【0154】

プログラム8101が処理される際、最初に処理func1が行われるとする。処理func1が呼び出され、処理func1がマッピングされている開始アドレスの命令がCPU8001によってフェッチされた瞬間に、バンク切り替え

装置 8 0 1 0 によってバンク切り替えのロックが実行される。このロックの実行によって、他の処理によるバンク切り替えが実行できないようになる。また、DSP 8 0 0 2 が処理を行っているかどうかの情報が使用状況監視装置 8 0 0 9 によって識別されていて、その情報はバンク切り替え装置 8 0 1 0 への入力信号となり、バンク切り替え装置 8 0 1 0 によってメモリ 8 0 0 6 へバンク切り替えを実行するための要求が行われている。

【0 1 5 5】

まず、DSP 8 0 0 2 が他の処理を行っていない場合、バンク切り替えは、表側が DSP 用メモリバンク 8 0 0 7 であり、裏側が CPU 用メモリバンク 8 0 0 8 である。先ほどのバンク切り替えのロックによって、このバンク切り替えの状態は維持される。バンク切り替えのロックが実行されている間は、バンク切り替えの表側が DSP 用メモリバンク 8 0 0 7 になっているので、CPU 8 0 0 1 は DSP 用メモリバンク 8 0 0 7 から処理 8 1 0 2 を取り出す。処理 8 1 0 2 には DSP 8 0 0 2 を使用するコマンドが含まれていて、CPU 8 0 0 1 によって、この DSP 8 0 0 2 を使用するコマンドが発行され、DSP 8 0 0 2 によって処理が行われる。DSP 8 0 0 2 は突き放し制御であるので、このコマンドの処理が終了する時間を特定することはできない。処理 func 1 の呼び出し元であるプログラム 8 1 0 1 の main 関数へ戻る際、この main 関数へ戻る命令が CPU 8 0 0 1 によってフェッチされた瞬間に、バンク切り替え装置 8 0 1 0 によってバンク切り替えのアンロックが実行される。そして、メモリ 8 0 0 6 へのバンク切り替え要求が受け付けられ、結果として、表側が CPU 用メモリバンク 8 0 0 8 となり、裏側が DSP 用メモリバンク 8 0 0 7 となる。

【0 1 5 6】

これとは逆の場合、つまり、処理 func 1 が呼び出された際、DSP 8 0 0 2 が他の処理を行っていた場合、バンク切り替えは、表側が CPU 用メモリバンク 8 0 0 8 であり、裏側が DSP 用メモリバンク 8 0 0 7 である。バンク切り替えのロックが実行されているので、このバンク切り替えの状態はアンロックが実行されるまで維持される。DSP 8 0 0 2 が他の処理のよって使用されているので、バンク切り替え装置 8 0 1 0 はメモリ 8 0 0 6 へ、バンク切り替えを表側が

CPU用メモリバンク8008、裏側がDSP用メモリバンク8007となるように要求することになる。

【0157】

バンク切り替えは表側がCPU用メモリバンク8008となっているので、CPU8001はCPU用メモリバンク8008から処理8105を取り出す。処理8105は、DSP8002を使用するコマンドを含んだ処理8102と同等の機能を持っていて、CPU8001を使用するコマンドを含んでいるので、CPU8001によって処理が行われる。処理8105の処理が終了し、処理func1の呼び出し元であるプログラム8101のmain関数へ戻る際、このmain関数へ戻る命令がCPU8001によってフェッチされた瞬間に、バンク切り替え装置8010によってバンク切り替えのアンロックが実行される。そして、メモリ8006へのバンク切り替え要求が受け付けられ、結果として、表側がDSP用メモリバンク8007となり、裏側がCPU用メモリバンク8008となる。CPU8001が処理8105を行っている間に、DSP8002を使用していた他の処理が終了した場合は、使用状況監視装置8009の出力を受けて、バンク切り替え装置8010が、バンク切り替えの表側をDSP用メモリバンク8007とし、裏側をCPU用メモリバンク8008とする要求を行うことになる。しかし、この要求が受け付けられてバンク切り替えが実行されるのは、前述と同様にバンク切り替えのアンロックが実行されてからである。

【0158】

次に、処理func1のDSP8002を使用する処理8102中のコマンドが、DSP8002によって処理が行われている間、次の処理である処理func2が呼び出されたとする。このとき、DSP8002は使用中であり、先ほどのバンク切り替えのアンロックはバンク切り替え装置8010によって実行されているので、バンク切り替えは表側がCPU用メモリバンク8008となり、裏側がDSP用メモリバンク8007となっている。処理func2が呼び出され、処理func2がマッピングされている開始アドレスの命令がCPU8001によってフェッチされた瞬間に、バンク切り替え装置8010によってバンク切り替えのロックが実行される。このロックの実行によって、他の処理によるバン

ク切り替えが実行できないようになる。

【0159】

バンク切り替えは表側がCPU用メモリバンク8008となっているので、CPU8001はCPU用メモリバンク8008から処理8106を取り出す。処理8106は、DSP8002を使用するコマンドを含んだ処理8103と同等の機能を持っていて、CPU8001を使用するコマンドを含んでいるので、CPU8001によって処理が行われる。処理8106の処理が終了し、処理func2の呼び出し元であるプログラム8101のmain関数へ戻る際、このmain関数へ戻る命令がCPU8001によってフェッチされた瞬間に、バンク切り替え装置8010によってバンク切り替えのアンロックが実行される。そして、メモリ8006へのバンク切り替え要求が受け付けられ、結果として、表側がDSP用メモリバンク8007となり、裏側がCPU用メモリバンク8008となる。CPU8001が処理8106を行っている間に、DSP8002を使用していた処理8102中のコマンドが終了した場合は、先ほどの処理func1の処理を行う際にDSP8002が使用中であった場合と同様に、バンク切り替え装置8010によってメモリ8006へバンク切り替えの要求が行われ、バンク切り替えのアンロックが実行されてから、この要求が受け付けられることになる。

【0160】

最後に処理func3が呼び出される。先ほどのDSP8002が使用中でない場合の処理func1の場合と同様に、バンク切り替え装置8010によって、バンク切り替えのロック、バンク切り替え要求が行われ、DSP用メモリバンク8007からDSP8002を使用するコマンドを含んだ処理8104がCPU8001によって取り出され、DSP8002を使用するコマンドは突き放し処理としてDSP8002によって実行される。プログラム8101のmain関数へ戻る際のバンク切り替えのアンロックの実行、バンク切り替えの表側をCPU用メモリバンク8008とし、裏側をDSP用メモリバンク8007とするバンク切り替えの実行についても同様である。

【0161】

以上のように本実施の形態によれば、バンク切り替え装置 8010 を用いた方法によって、DSP 8002 における DSP 競合の発生を軽減し、DSP 8002 を効率良く使用することが可能となる。この方法を実現するために、プログラム 8101 を修正する必要がなくそのまま適用できるので、ソフトウェアのオーバーヘッドがほとんどない。また、使用状況監視装置 8009 およびバンク切り替え装置 8010 という専用のハードウェアを用いることによって、バス 8003 の使用状況の監視およびソフトウェア処理変更を比較的高速に実現することができる。

【0162】

なお、本実施の形態では、プロセッサを 1 つの CPU、リソースを 1 つの DSP としたが、それぞれ複数の CPU、DSP でも良く、少なくとも 1 つのリソースとそのリソースを使用する少なくとも 1 つのプロセッサであれば良いものである。

【0163】

【発明の効果】

以上説明したように、本発明によれば、リソースの競合を動的に判定し、リソースを使用する処理の処理方法を変更することでリソース競合時の処理の停止を避け、システムの処理速度の低下を抑制することができる。

【0164】

その第 1 の方法としては、ソフトウェアをコンパイルする段階で、リソースを使用する処理を識別し、その処理に対しリソース競合を判定する処理と、その判定結果に基づいて処理を別の等価処理に置き換える一連の処理を組み込む。これにより、リソース競合を避け、システムの処理速度の低下を避けることができる。

【0165】

さらに、リソースが共有メモリに接続されたバスの場合、バス競合の履歴を採取することで、さらに高精度なリソース競合の判定が可能である。

【0166】

また、第 2 の方法としては、プロセッサで任意の処理を実行中に新規の処理要

求が発生した場合、割り込み信号をプロセッサへ与えることで新規の処理を別プロセッサで代用させる。本発明を実現するために割り込みルーチンを用意する以外は、もともとのソフトウェアに対して処理の変更、追加をする必要が無く、ソフトウェアをそのまま適用でき、さらにソフトウェアによる競合対策の処理がないため、ソフトウェアのオーバーヘッドが少ない。

【0167】

また、第3の方法として、あらかじめ同一アドレスを割り当てた複数のメモリバンクを用意し、プロセッサが任意の処理を実行中に新規の処理要求が発生した場合、メモリバンクを切り替えることで新規処理を別のプロセッサで代わりに実行させる。これについては、競合対策の処理がメモリバンク切り替え回路のみで実現できるため、他の方法に比べて高速である。

【0168】

また、第4の方法として、処理の待ち時間や処理時間などを計測することでリソース競合が発生中か否かを判定し、リソース競合が発生中ならば新たにリソース競合を引き起こす処理を発行しない。これにより、競合によるシステムの処理速度の低下を防ぐことが可能である。

【0169】

さらに、第4の方法に対し、定期的または不定期に競合判定を補正することで、さらに高精度な競合判定が可能である。

【0170】

さらに、第4の方法に対し、競合情報を競合を引き起こす処理の出現箇所とともに記憶することで、ループ処理における競合判定を高精度に行うことが可能である。

【図面の簡単な説明】

【図1】 本発明の実施の形態におけるソフトウェア処理システムの基本構成を示すブロック図

【図2】 本発明の第1の実施の形態におけるソフトウェア処理システムのハードウェアの構成を示すブロック図

【図3】 本発明の第1の実施の形態におけるソフトウェア処理システムのバ

スの使用状況監視装置の構成を示す図

【図 4】 本発明の第 1 の実施の形態におけるソフトウェア処理システムのソフトウェアの構成を示す図

【図 5】 本発明の第 1 の実施の形態におけるソフトウェア処理システムにおけるコンパイルフローを示すフローチャート

【図 6】 本発明の第 1 の実施の形態におけるソフトウェア処理システムにおけるバスアクセス識別装置が識別するバスアクセスの状態を示す図

【図 7】 本発明の第 1 の実施の形態におけるソフトウェア処理システムにおける処理付加手段によるソフトウェアの変更を示す図

【図 8】 本発明の第 2 の実施の形態におけるソフトウェア処理システムのハードウェアの構成を示すブロック図

【図 9】 本発明の第 2 の実施の形態におけるソフトウェア処理システムのバス調停回路の制御フローを示すフローチャート

【図 10】 本発明の第 2 の実施の形態におけるソフトウェア処理システムのコンパイラの処理フローを示すフローチャート

【図 11】 本発明の第 2 の実施の形態におけるソフトウェア処理システムの外部メモリへ頻繁にアクセスする処理に対する変更を示す図

【図 12】 本発明の第 2 の実施の形態におけるソフトウェア処理システムのバス競合の判定フローを示すフローチャート

【図 13】 本発明の第 2 の実施の形態におけるソフトウェア処理システムの等価処理への置換を行う確率を示す図

【図 14】 本発明の第 2 の実施の形態におけるソフトウェア処理システムの出現箇所を考慮した場合のバス競合の判定フローを示すフローチャート

【図 15】 本発明の第 2 の実施の形態におけるソフトウェア処理システムのバス競合の判定フローを示すフローチャート

【図 16】 本発明の第 3 の実施の形態におけるソフトウェア処理システムのハードウェア構成を示すブロック図

【図 17】 本発明の第 3 の実施の形態におけるソフトウェア処理システムのコンパイラの処理フローを示すフローチャート

【図 18】 本発明の第 3 の実施の形態におけるソフトウェア処理システムのコンパイラによる処理の変更を示す図

【図 19】 本発明の第 3 の実施の形態におけるソフトウェア処理システムの DSP 競合の判定フローを示すフローチャート

【図 20】 本発明の第 3 の実施の形態におけるソフトウェア処理システムの処理の流れを示すシーケンス図

【図 21】 本発明の第 4 の実施の形態におけるソフトウェア処理システムのハードウェア構成を示すブロック図

【図 22】 本発明の第 4 の実施の形態におけるソフトウェア処理システムの調停回路の制御フローを示すフローチャート

【図 23】 本発明の第 4 の実施の形態におけるソフトウェア処理システムのコンパイラの処理フローを示すフローチャート

【図 24】 本発明の第 4 の実施の形態におけるソフトウェア処理システムの DSP で実行する処理に対する変更を示す図

【図 25】 本発明の第 4 の実施の形態におけるソフトウェア処理システムの DSP 競合の判定フローを示すフローチャート

【図 26】 本発明の第 4 の実施の形態におけるソフトウェア処理システムの等価処理への置換を行う確率を示す図

【図 27】 本発明の第 4 の実施の形態におけるソフトウェア処理システムの出現箇所を考慮した場合の DSP 競合の判定フローを示すフローチャート

【図 28】 本発明の第 4 の実施の形態におけるソフトウェア処理システムの DSP 競合の判定フローを示すフローチャート

【図 29】 本発明の第 5 の実施の形態におけるソフトウェア処理システムのハードウェア構成を示すブロック図

【図 30】 本発明の第 5 の実施の形態におけるソフトウェア処理システムのソフトウェアを示す図

【図 31】 本発明の第 5 の実施の形態におけるソフトウェア処理システムの DSP 競合時の処理フローを示すフローチャート

【図 32】 本発明の第 5 の実施の形態におけるソフトウェア処理システムに

おける処理の流れを示すシーケンス図

【図 3 3】 本発明の第 6 の実施の形態におけるソフトウェア処理システムのハードウェア構成を示すブロック図

【図 3 4】 本発明の第 6 の実施の形態におけるソフトウェア処理システムのソフトウェアの構成を示す図

【図 3 5】 本発明の第 6 の実施の形態におけるソフトウェア処理システムのコンパイルフローを示すフローチャート

【図 3 6】 本発明の第 6 の実施の形態におけるソフトウェア処理システムのメモリバンクマッピングを示す図

【符号の説明】

- 101 プロセッサ
- 102 リソース
- 103 使用状況監視処理手段
- 104 ソフトウェア処理変更処理手段
- 1001 CPU
- 1002 バス
- 1003 周辺回路
- 1004 外部メモリ
- 1005 使用状況監視処理装置
- 1101 バスアクセス識別装置
- 1102 バスアクセス状態レジスタ
- 1201 プログラム
- 1301 コンパイル手段
- 1302 処理識別手段
- 1303 処理付加手段
- 2001 プロセッサ
- 2002 DSP
- 2003 記憶装置
- 2004 周辺回路

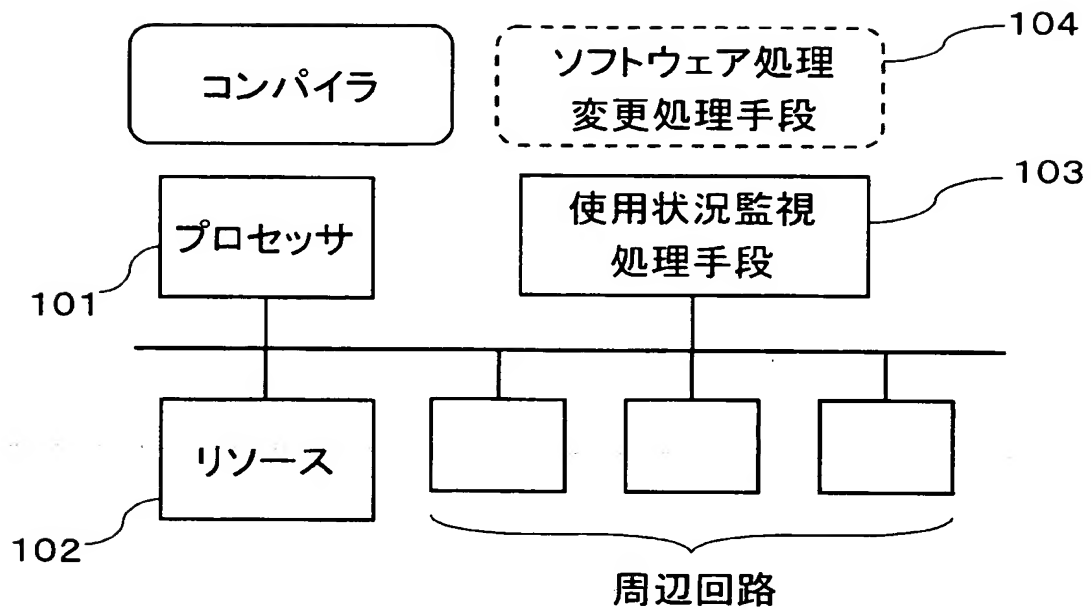
- 2 0 0 5 バス調停回路
- 2 0 0 6 外部メモリ
- 2 0 0 7 共有バス
- 2 1 0 1 アクセス要求判定手段
- 2 1 0 2 メモリ状態判定手段
- 2 1 0 3 メモリアクセス許可手段
- 2 1 0 4 処理終了判定手段
- 2 2 0 1 処理識別手段
- 4 0 0 1 D S P
- 4 0 0 2 プロセッサ
- 4 0 0 3 D S P 使用状況監視装置
- 4 0 0 4 メモリ
- 4 1 0 1 コンパイラ
- 4 1 0 2 ソースコード
- 4 1 0 3 プロセッサ用ライブラリ
- 4 1 0 4 D S P 用ライブラリ
- 4 1 0 5 オブジェクトコード
- 4 1 0 6 プロセッサ用ライブラリ
- 4 1 0 7 D S P 用ライブラリ
- 5 0 0 1 プロセッサ
- 5 0 0 2 D S P
- 5 0 0 3 記憶装置
- 5 0 0 4 周辺回路
- 5 0 0 5 外部メモリ
- 5 0 0 6 共有バス
- 5 0 0 7 バス調停回路
- 5 0 0 8 調停回路
- 5 1 0 1 処理要求判定手段
- 5 1 0 2 D S P 状態判定手段

- 5 1 0 3 D S P 使用許可手段
- 5 1 0 4 処理終了判定手段
- 5 2 0 1 処理識別手段
- 7 0 0 1 D S P
- 7 0 0 2 プロセッサ
- 7 0 0 3 D S P 使用状況監視装置
- 7 0 0 4 メモリ
- 8 0 0 1 C P U
- 8 0 0 2 D S P
- 8 0 0 3 バス
- 8 0 0 4 外部メモリ
- 8 0 0 5 周辺回路
- 8 0 0 6 メモリ
- 8 0 0 7 D S P 用メモリバンク
- 8 0 0 8 C P U 用メモリバンク
- 8 0 0 9 使用状況監視装置
- 8 0 1 0 バンク切り替え装置
- 8 1 0 1 プログラム
- 8 2 0 1 コンパイル手段
- 8 2 0 2 処理識別手段
- 8 2 0 3 マッピング手段

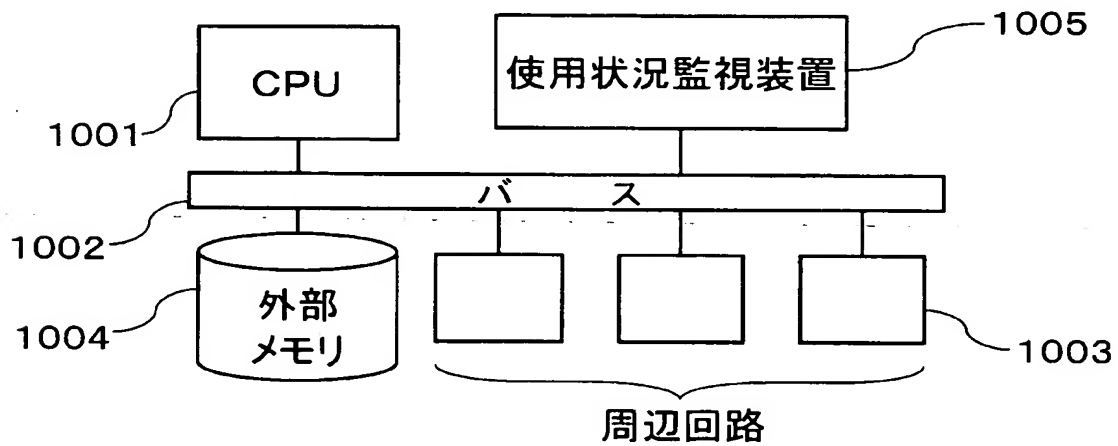
【書類名】

図面

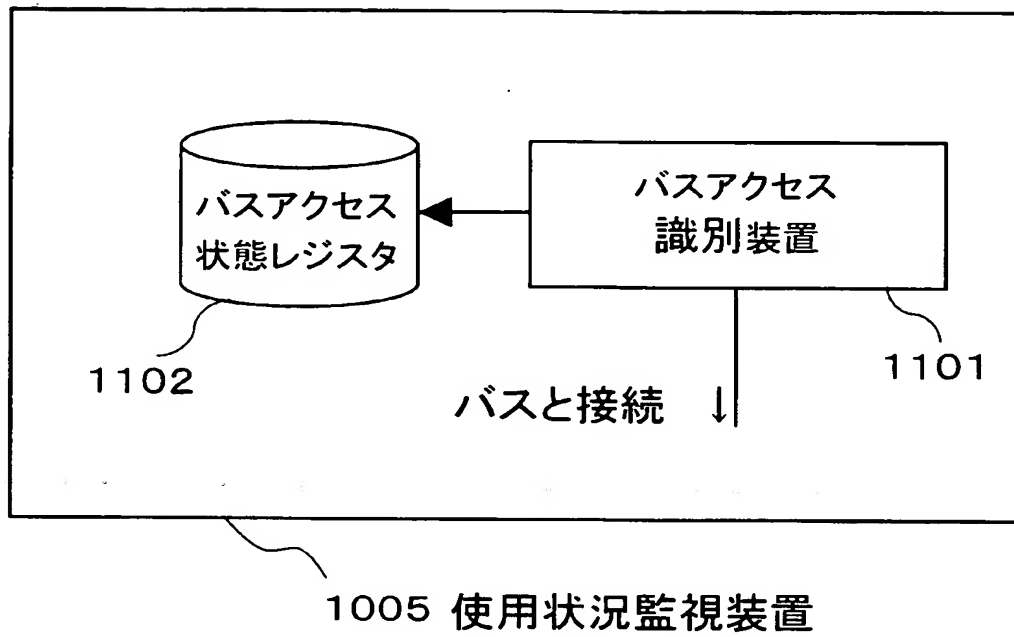
【図 1】



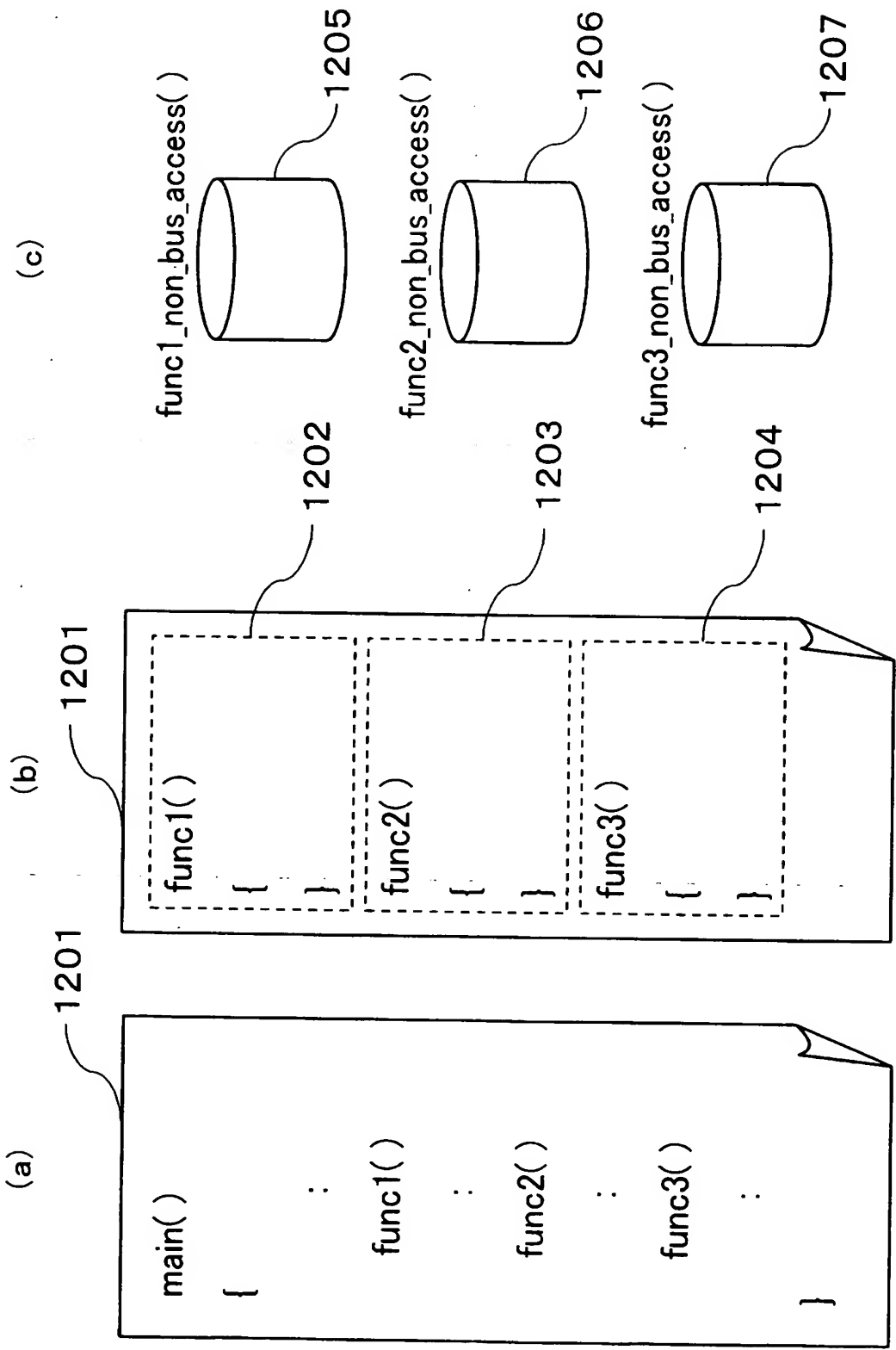
【図 2】



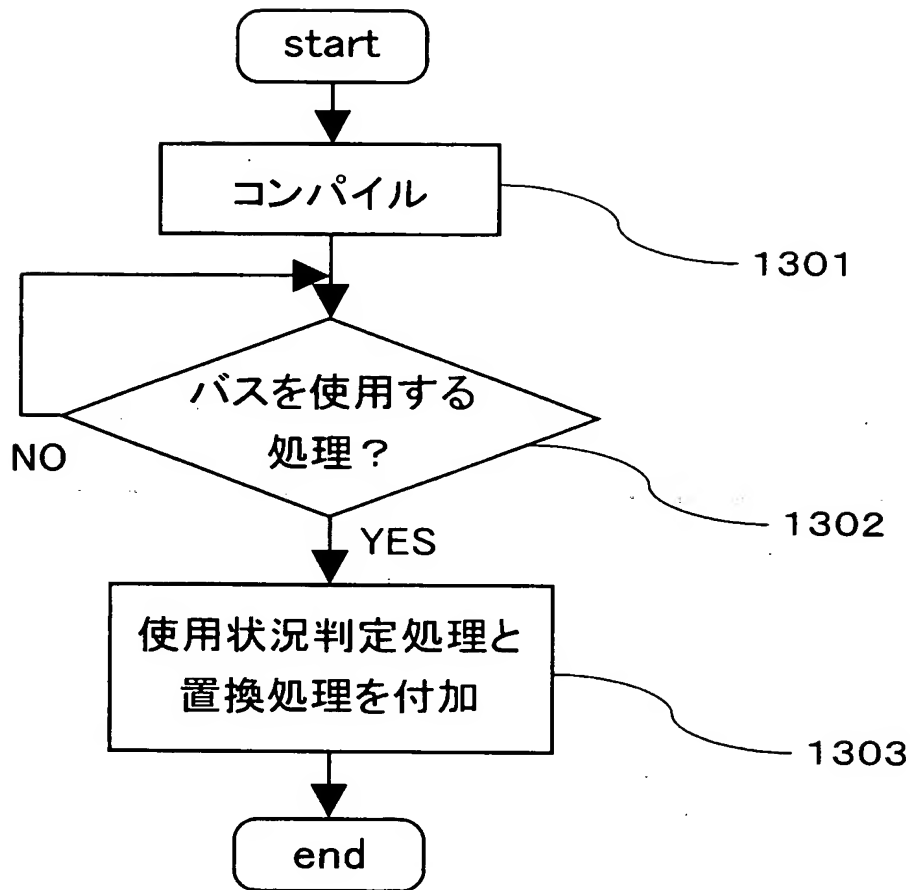
【図 3】



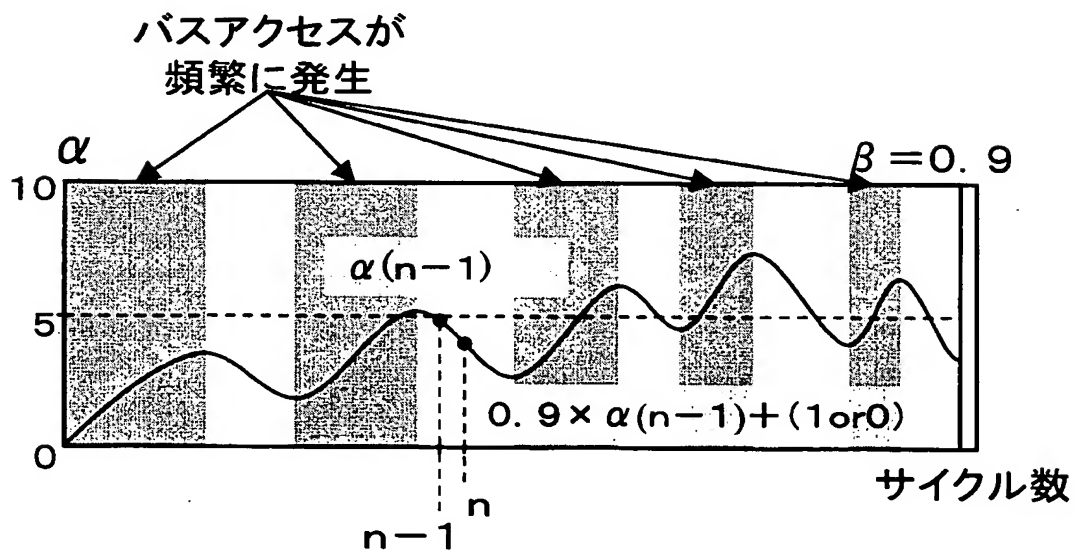
【図 4】



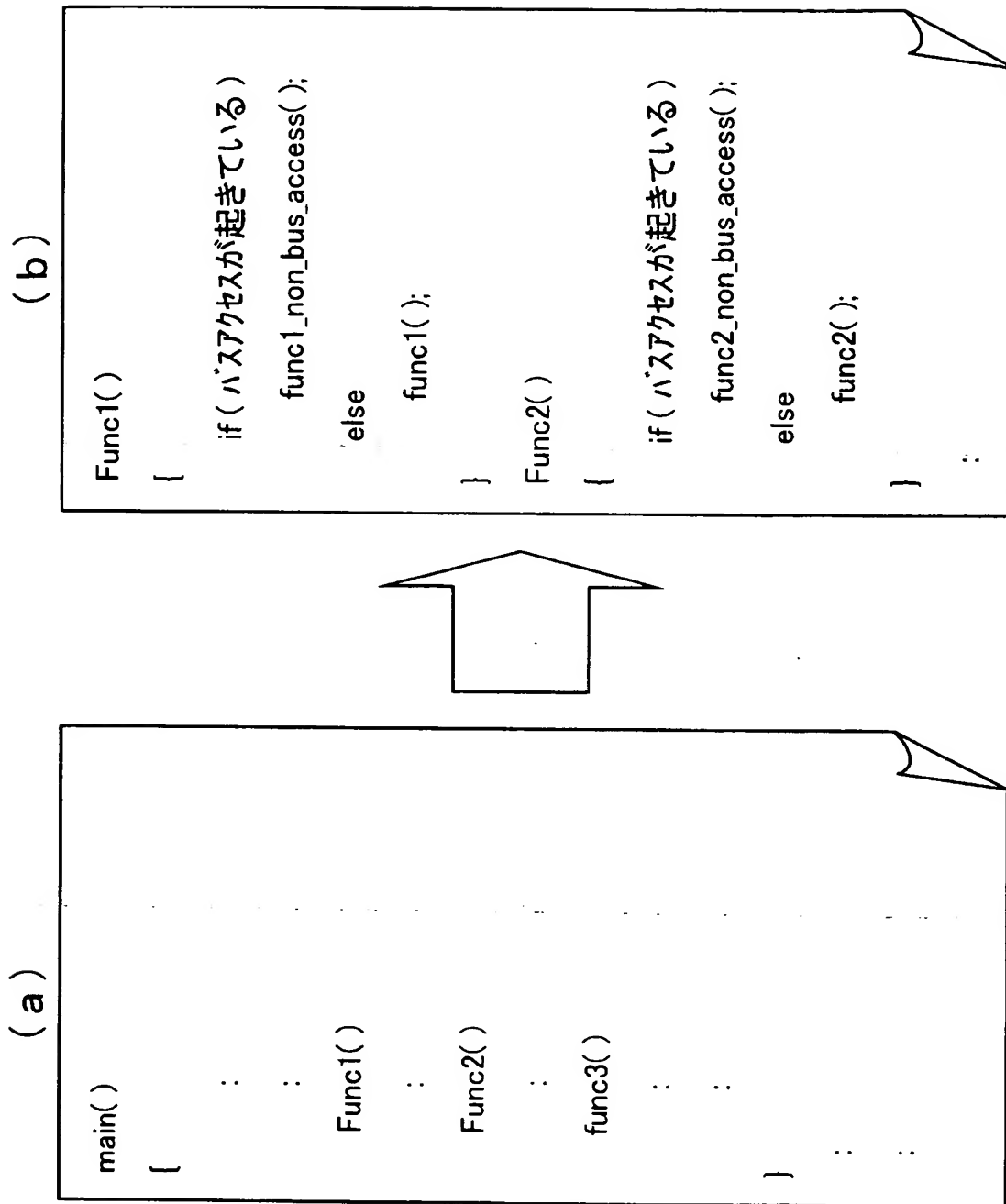
【図5】



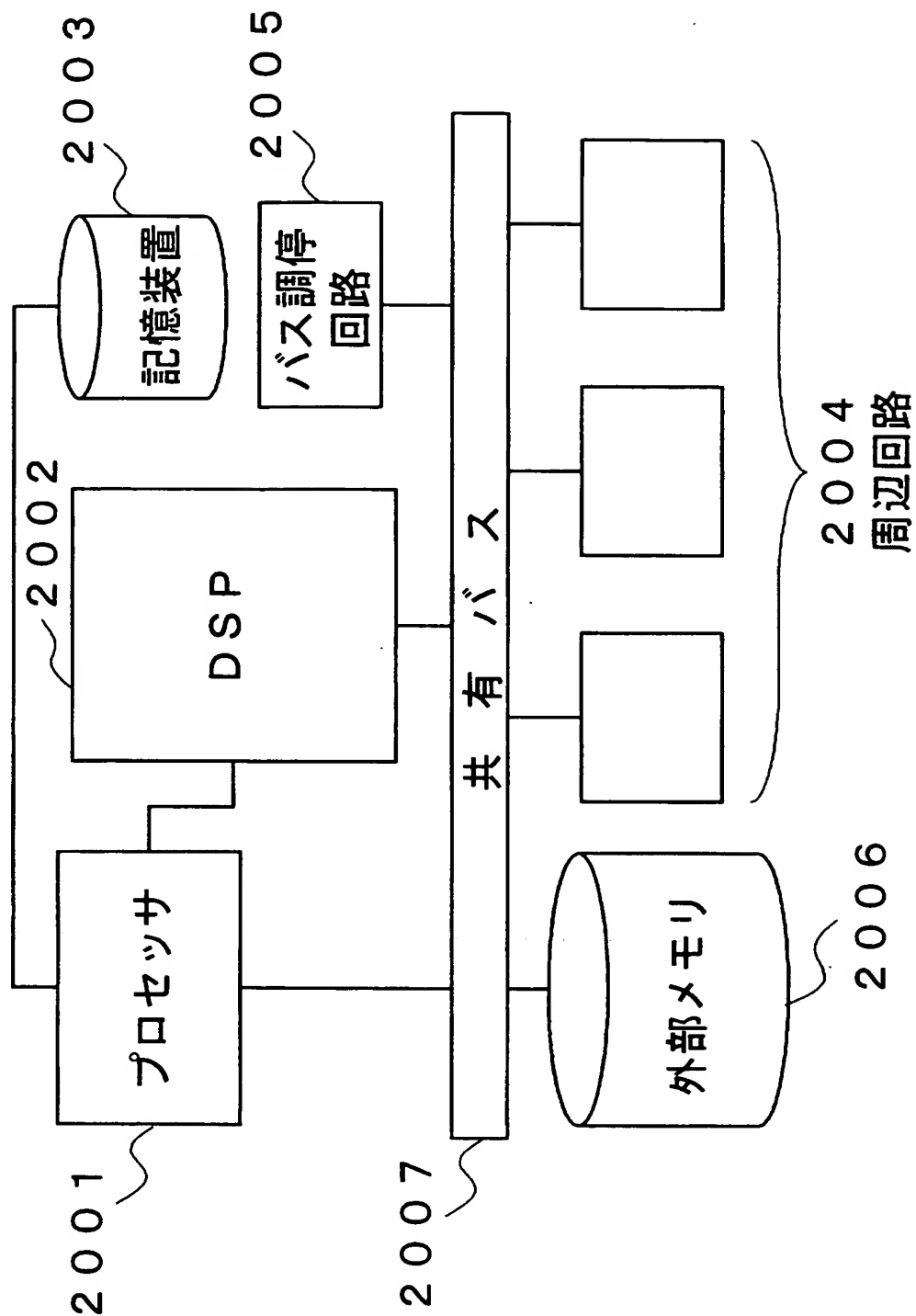
【図6】



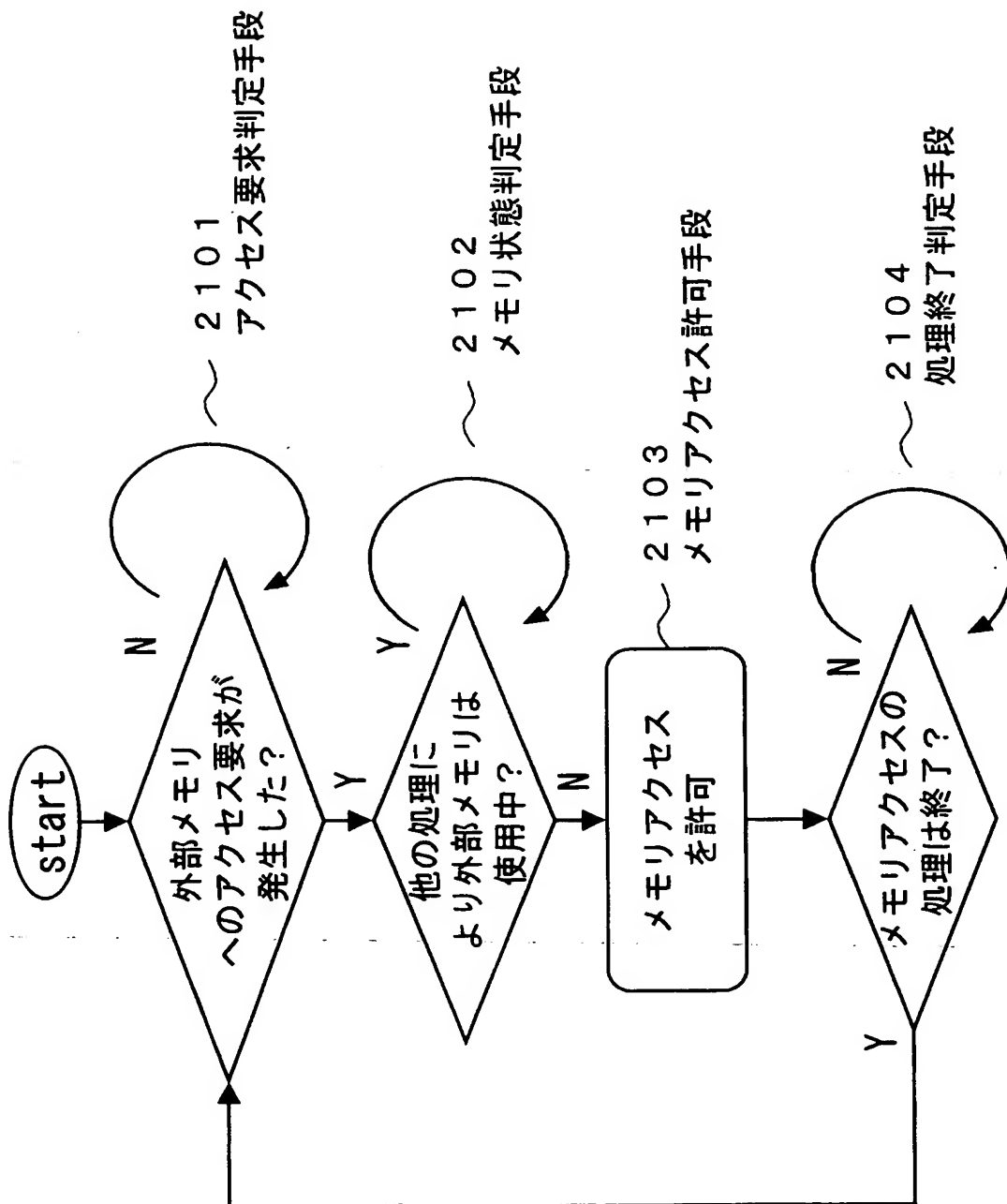
【図 7】



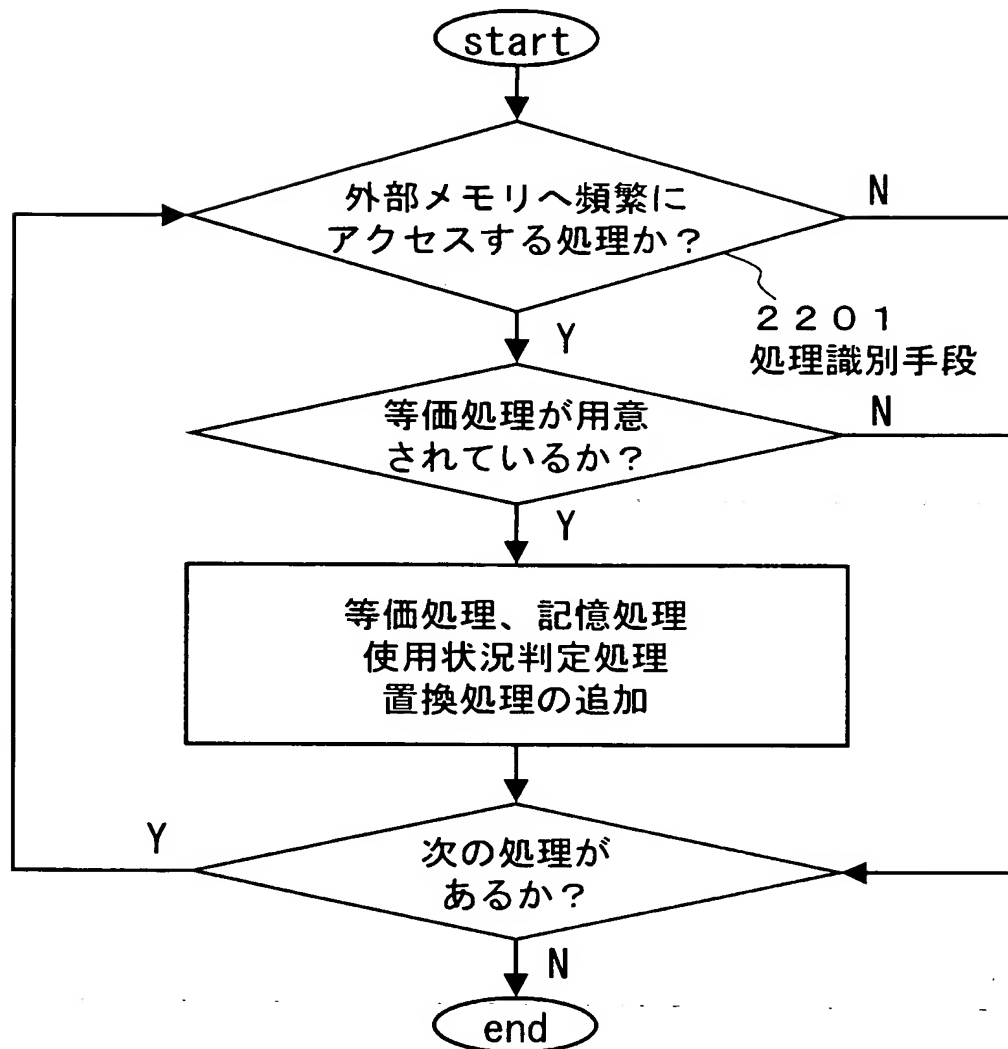
【図 8】



【図 9】



【図10】



【図 11】

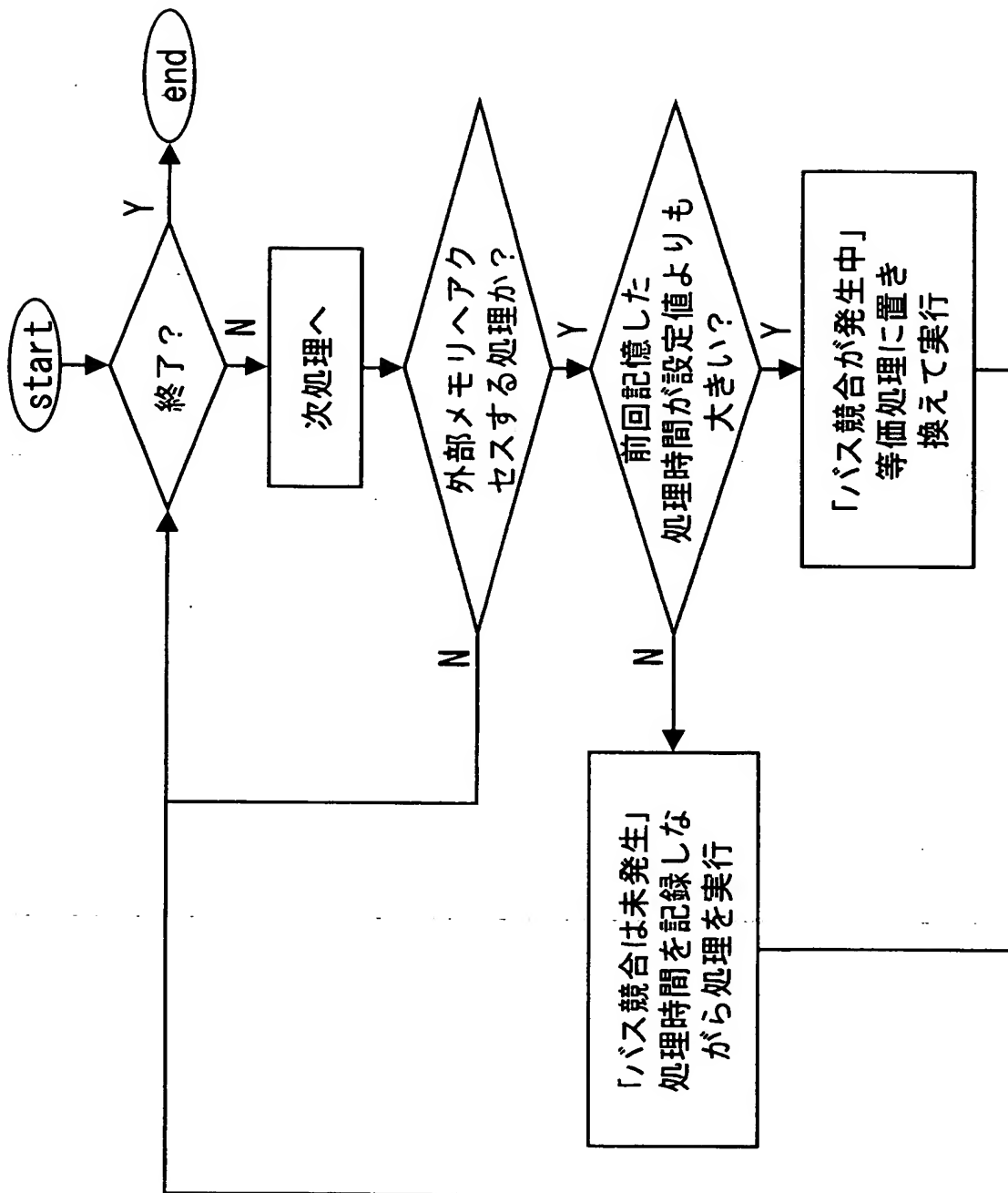
```
1: proc_task1() {  
2:     mem_acs_many();  
3: };
```

(a) コンパイラ実行前

```
1: proc_task1() {  
2:     if( task_time < DEFINED_TIME ) {  
3:         start_time = timer_count;  
4:         mem_acs_many();  
5:         end_time = timer_count;  
6:         task_time = end_time - start_time;  
7:     }  
8:     else {  
9:         mem_acs_few();  
10:    };  
11: };
```

(b) コンパイラ実行後

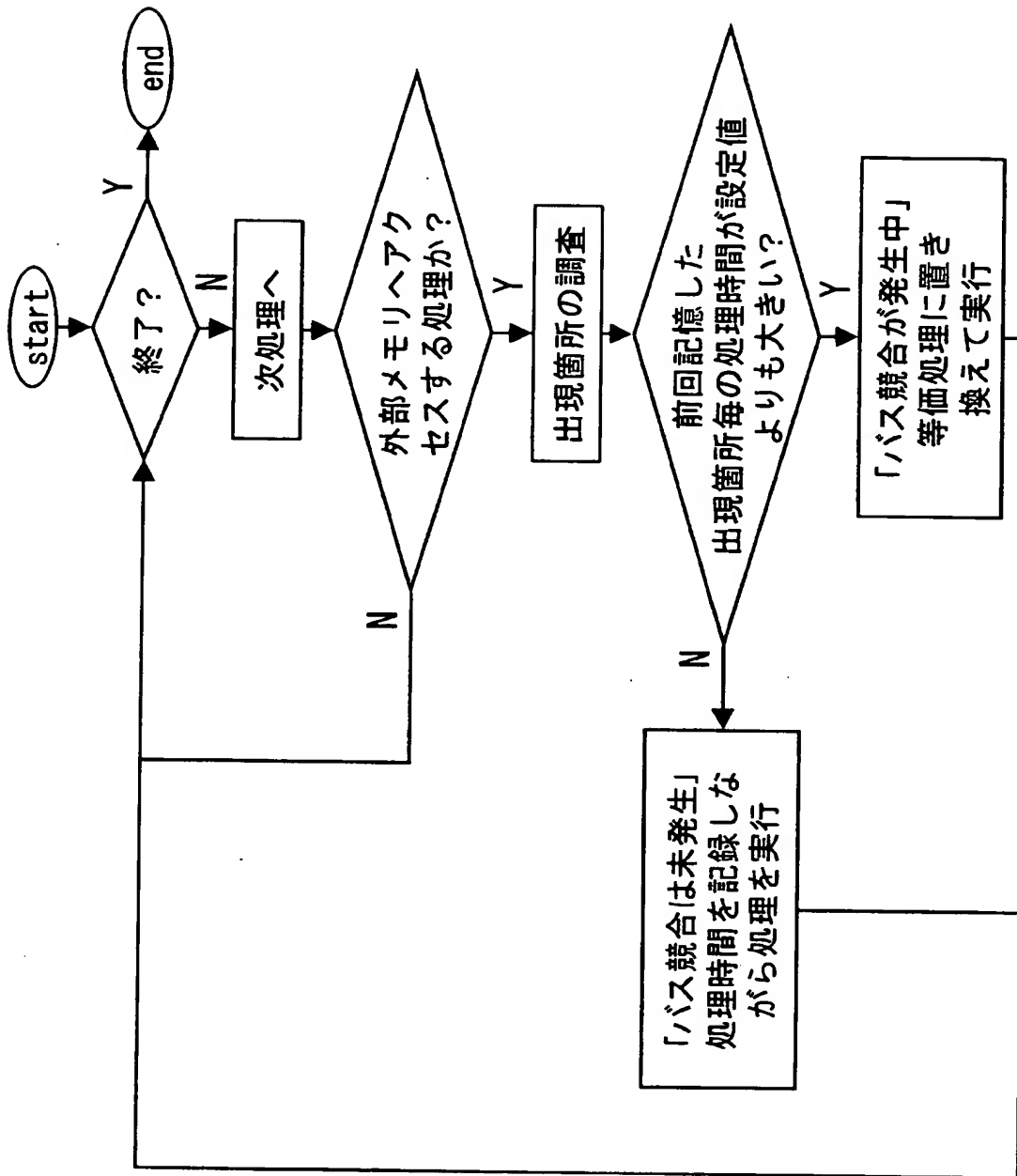
【図 12】



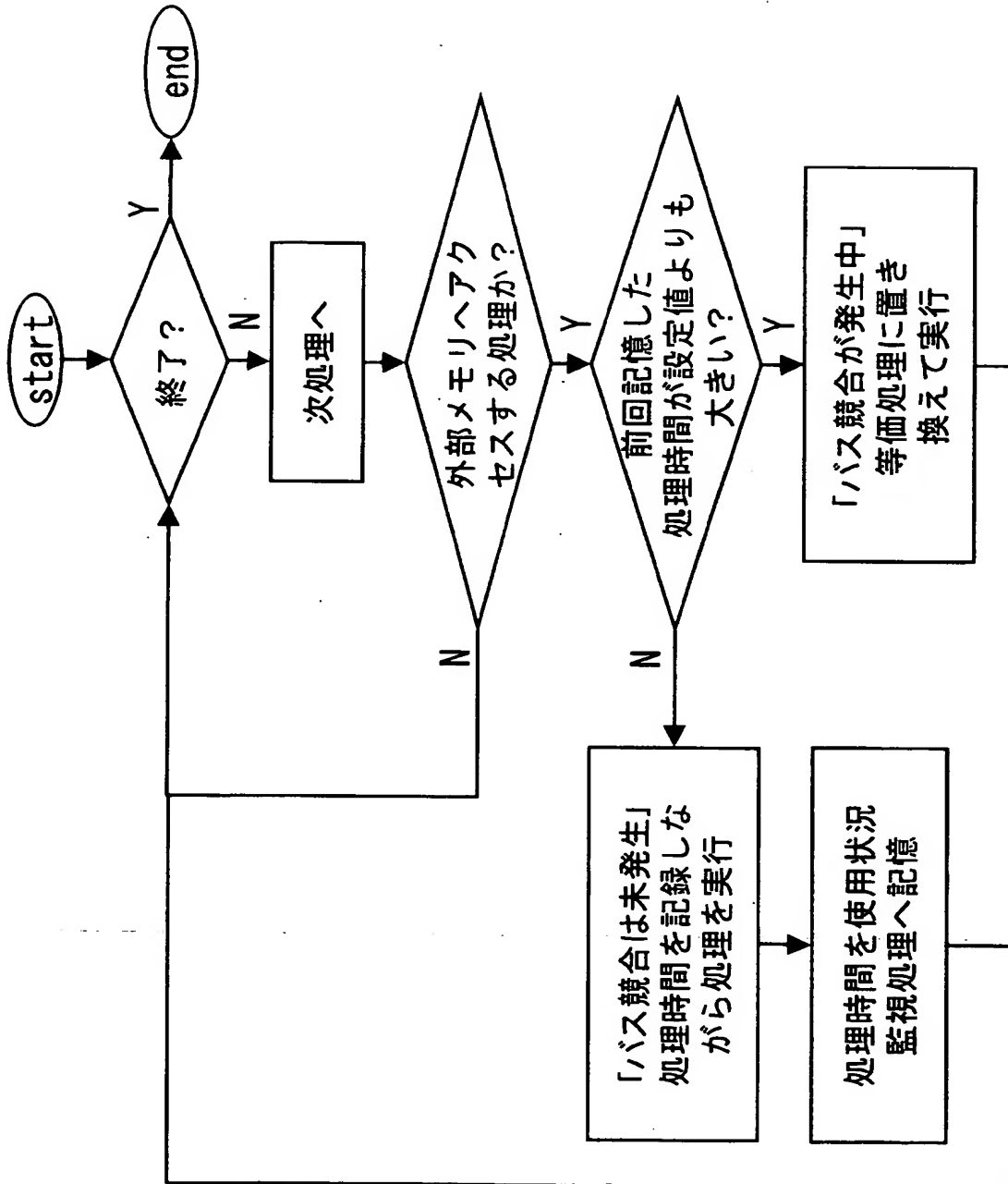
【図 1 3】

処理時間 設定値	等価処理へ置き 換える確率[%]	等価処理へ置き 換えない確率[%]
~ 2. 0	60	40
2. 0 ~ 1. 8	70	30
1. 8 ~ 1. 6	80	20
1. 6 ~ 1. 4	90	10
1. 4 ~ 1. 2	95	5
1 以下	0	100

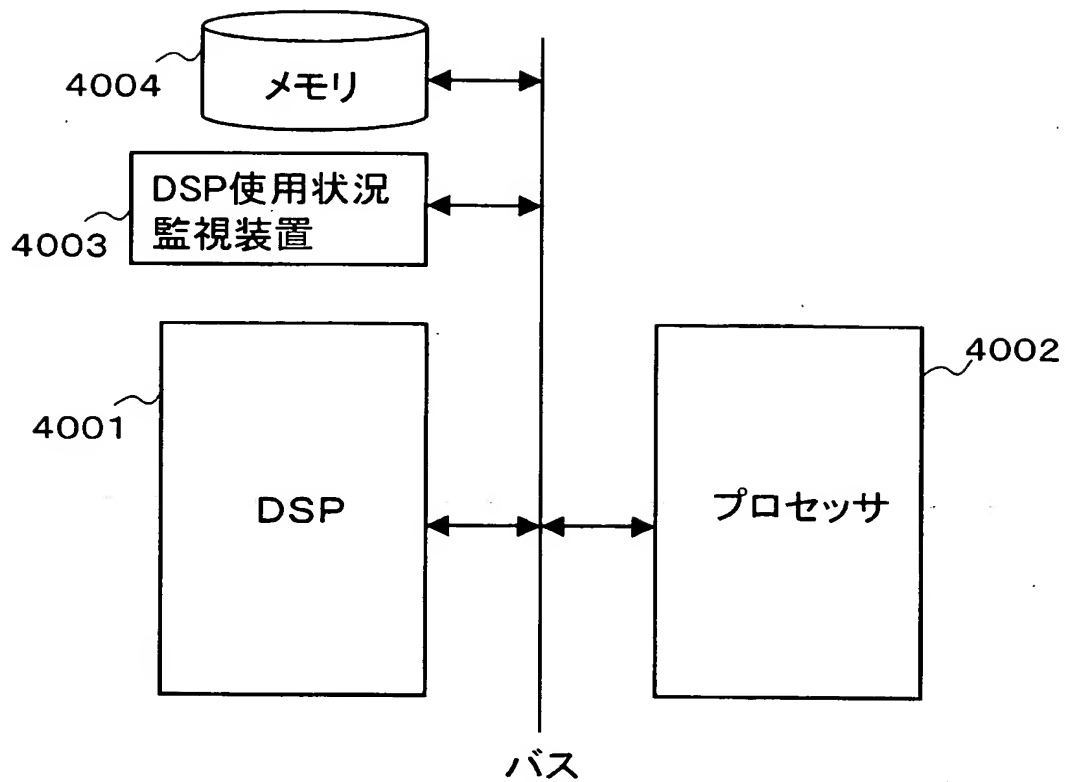
【図 14】



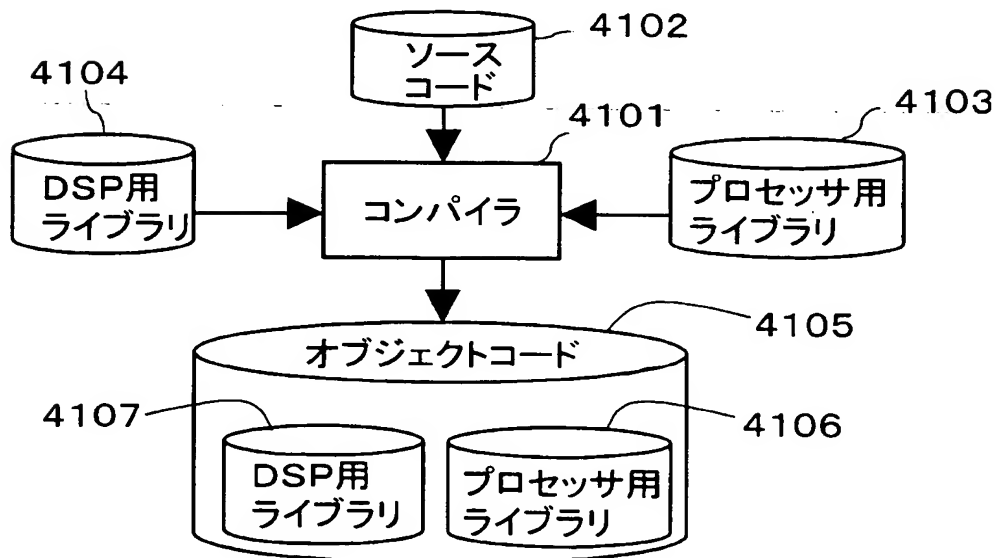
【図 15】



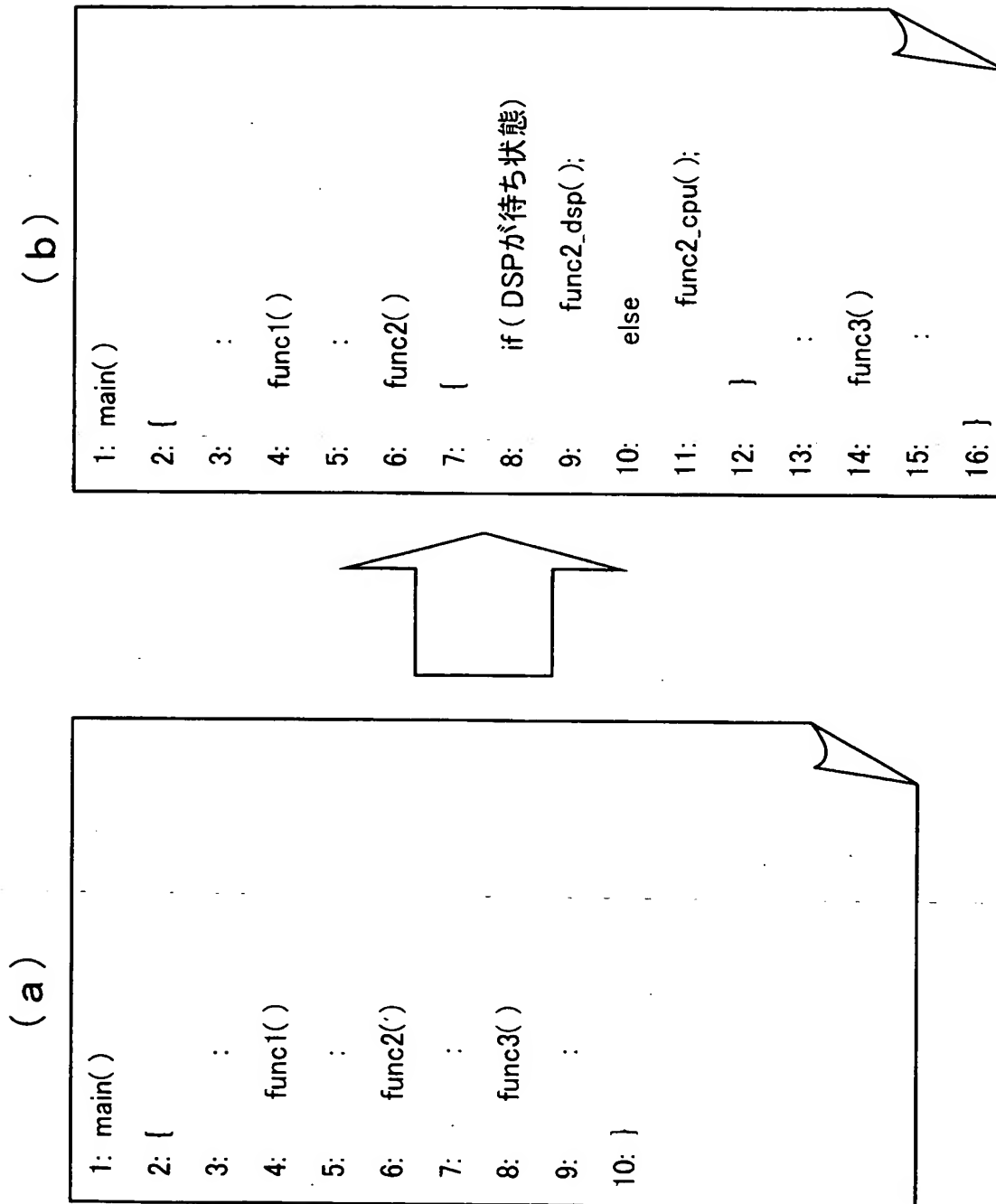
【図 16】



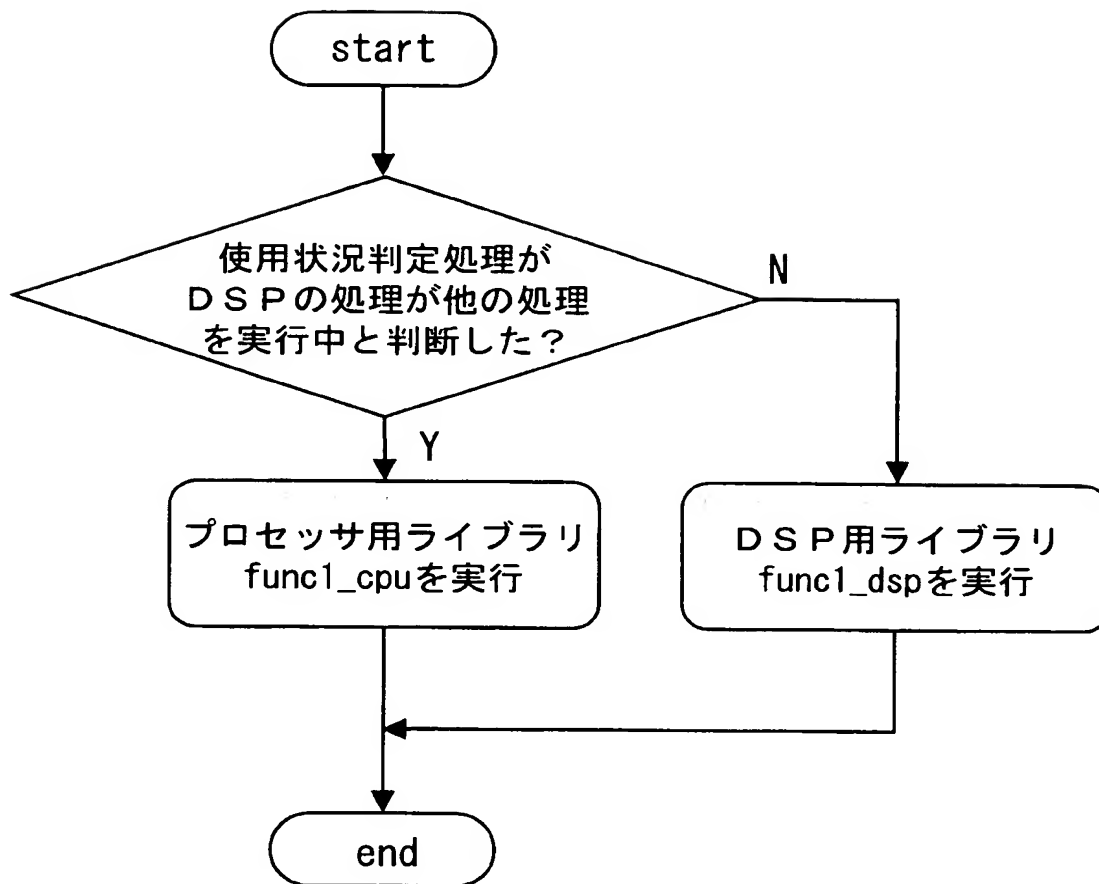
【図 17】



【図 18】

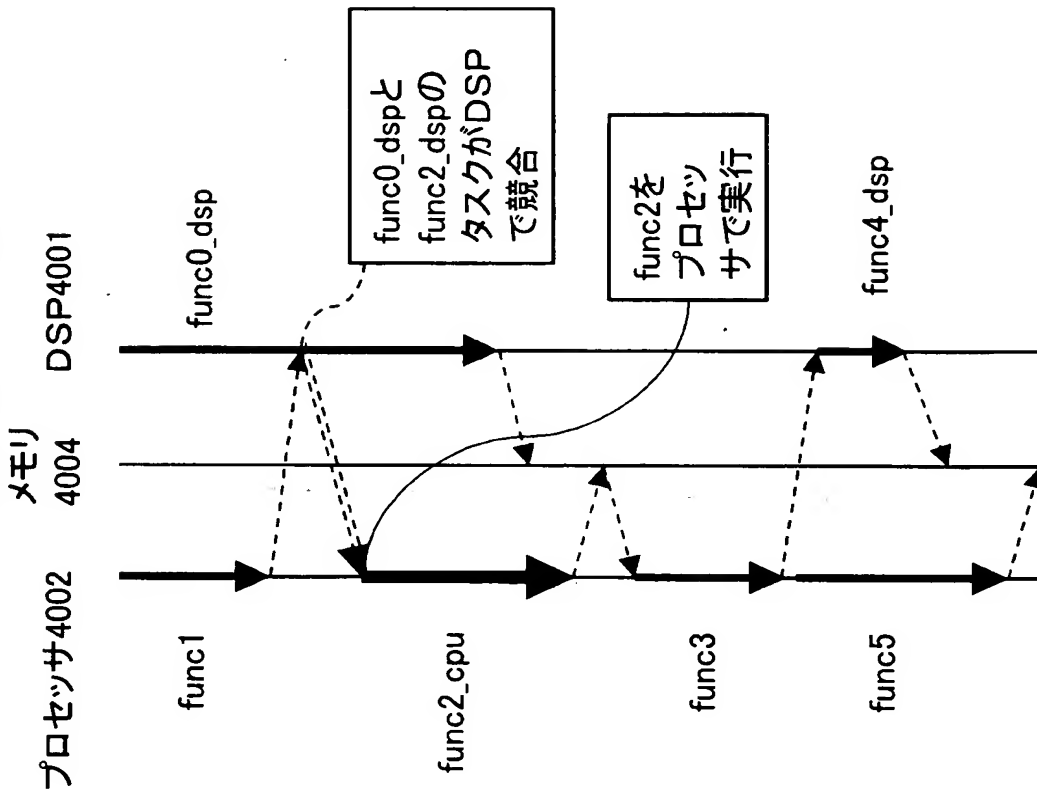


【図 19】

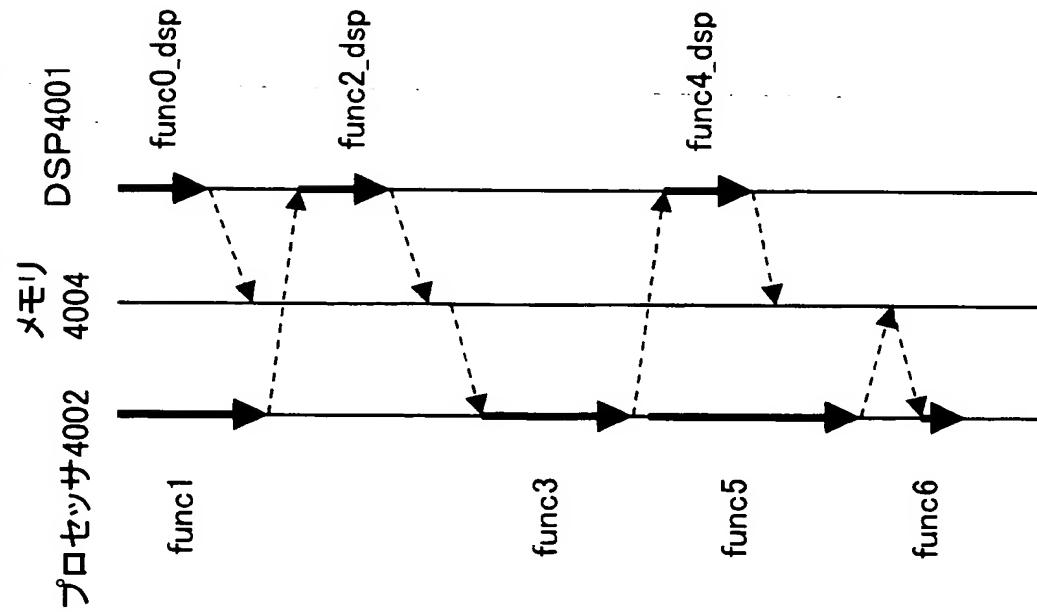


【図 20】

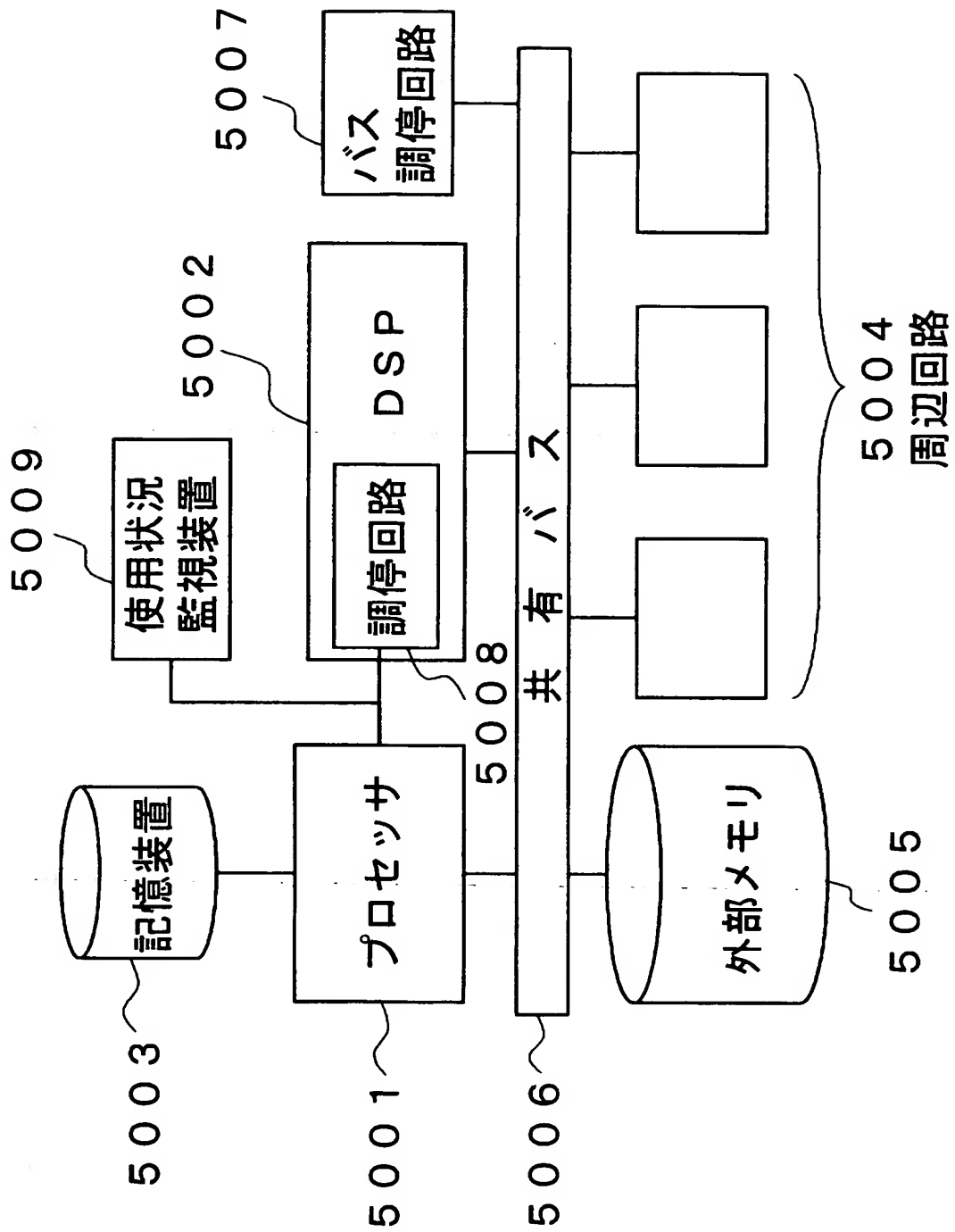
(b) DSPのタスクが競合した場合



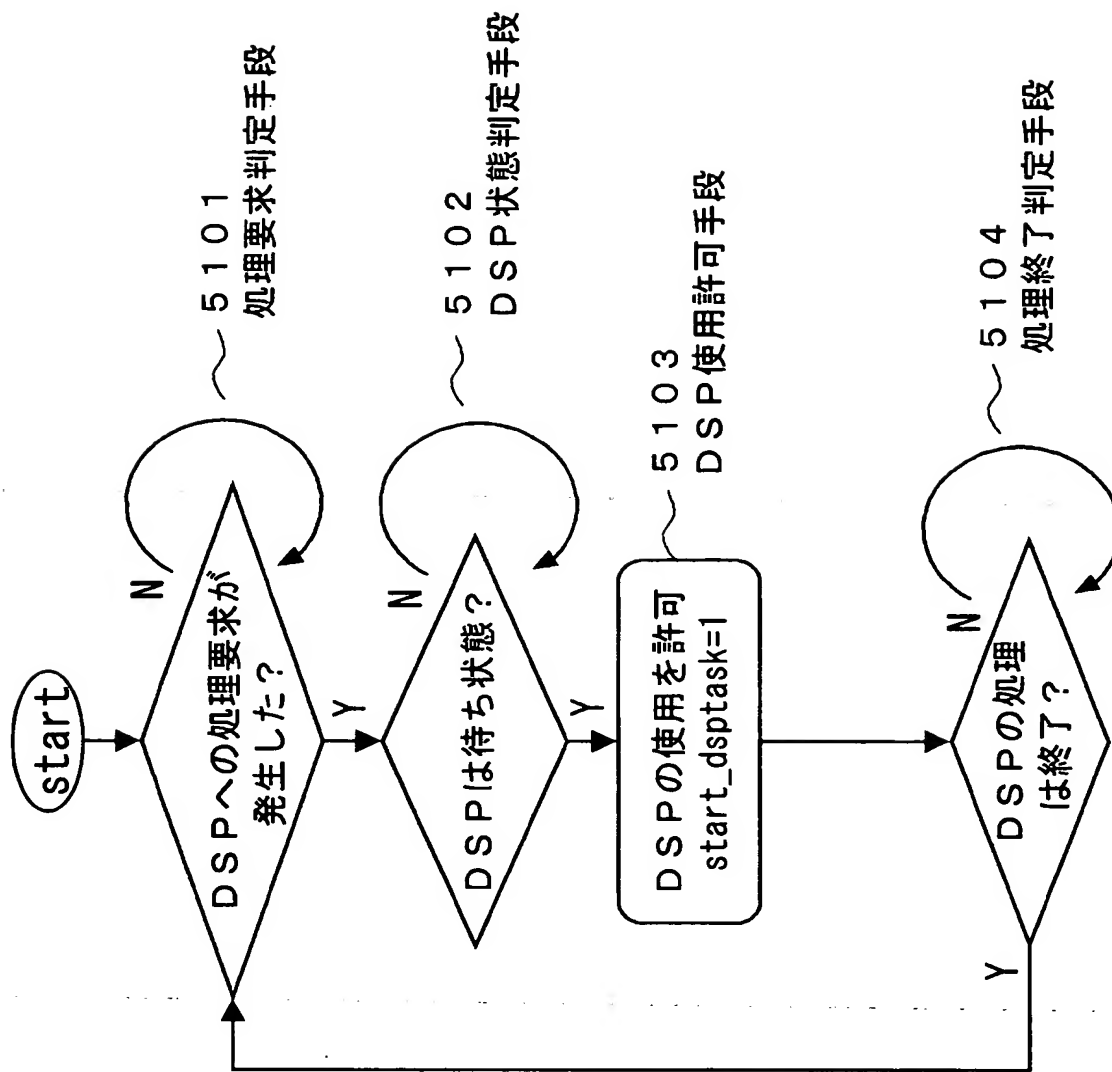
(a) DSPのタスクが競合しない場合



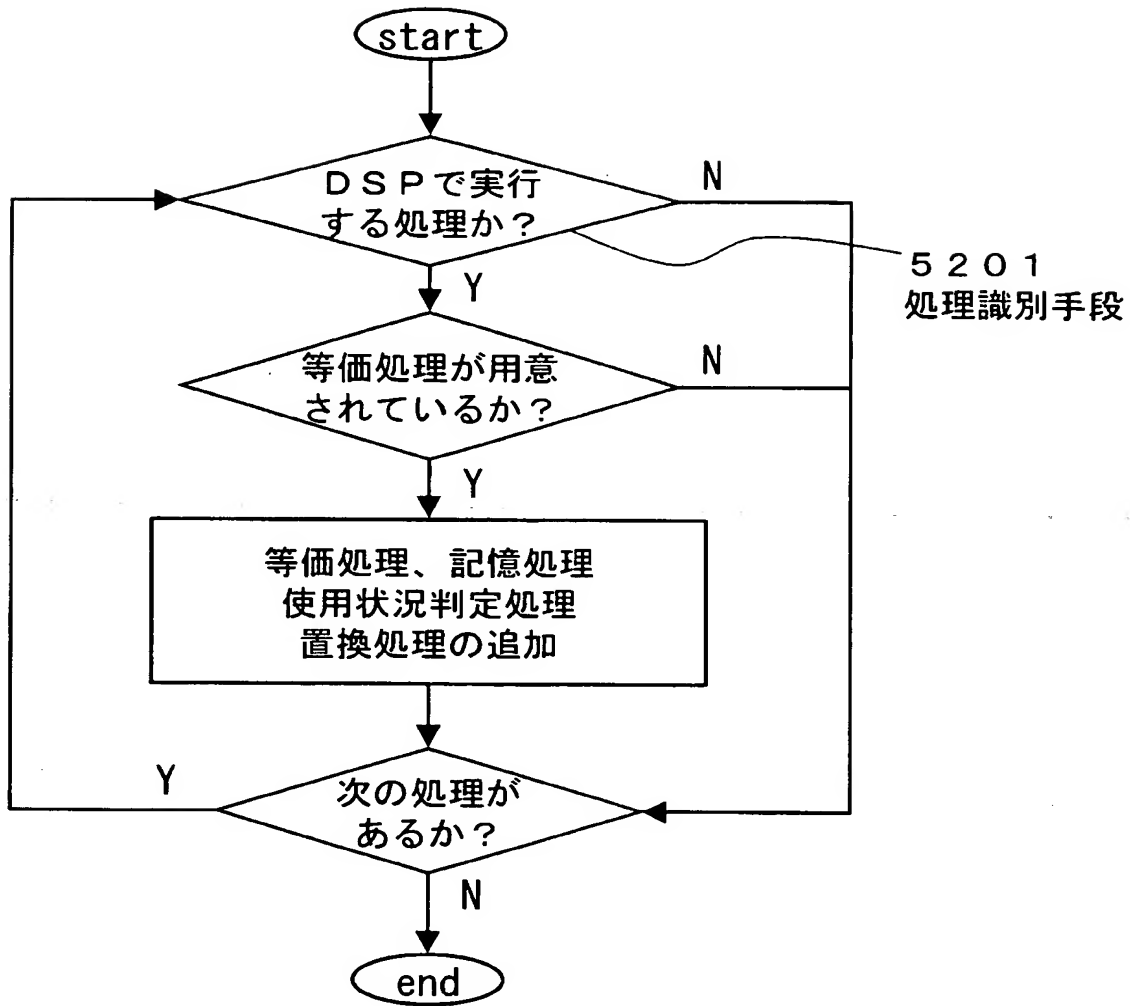
【図 21】



【図 22】



【図 23】



【図 24】

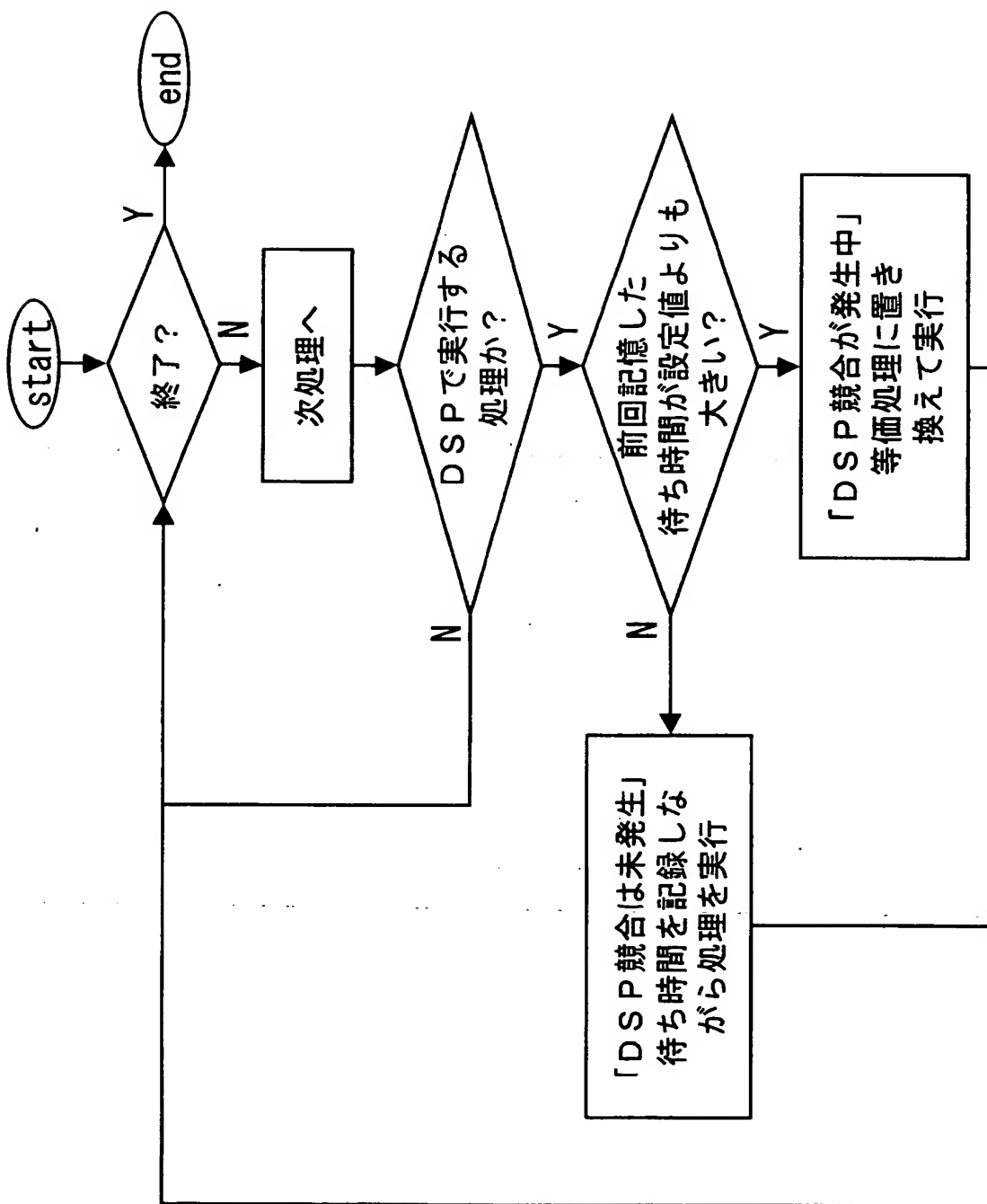
```
1: proc_task1() {  
2:     dsp_task();  
3: };
```

(a) コンパイラ実行前

```
1: proc_task1() {  
2:     if( wait_time < DEFINED_TIME ) {  
3:         start_time = timer_count;  
4:         (DSPが処理を開始できるまで待つ)  
5:         end_time = timer_count;  
6:         wait_time = end_time - start_time;  
7:         dsp_task();  
8:     }  
9:     else {  
10:         proc_dsptask();  
11:     };  
12: };
```

(b) コンパイラ実行後

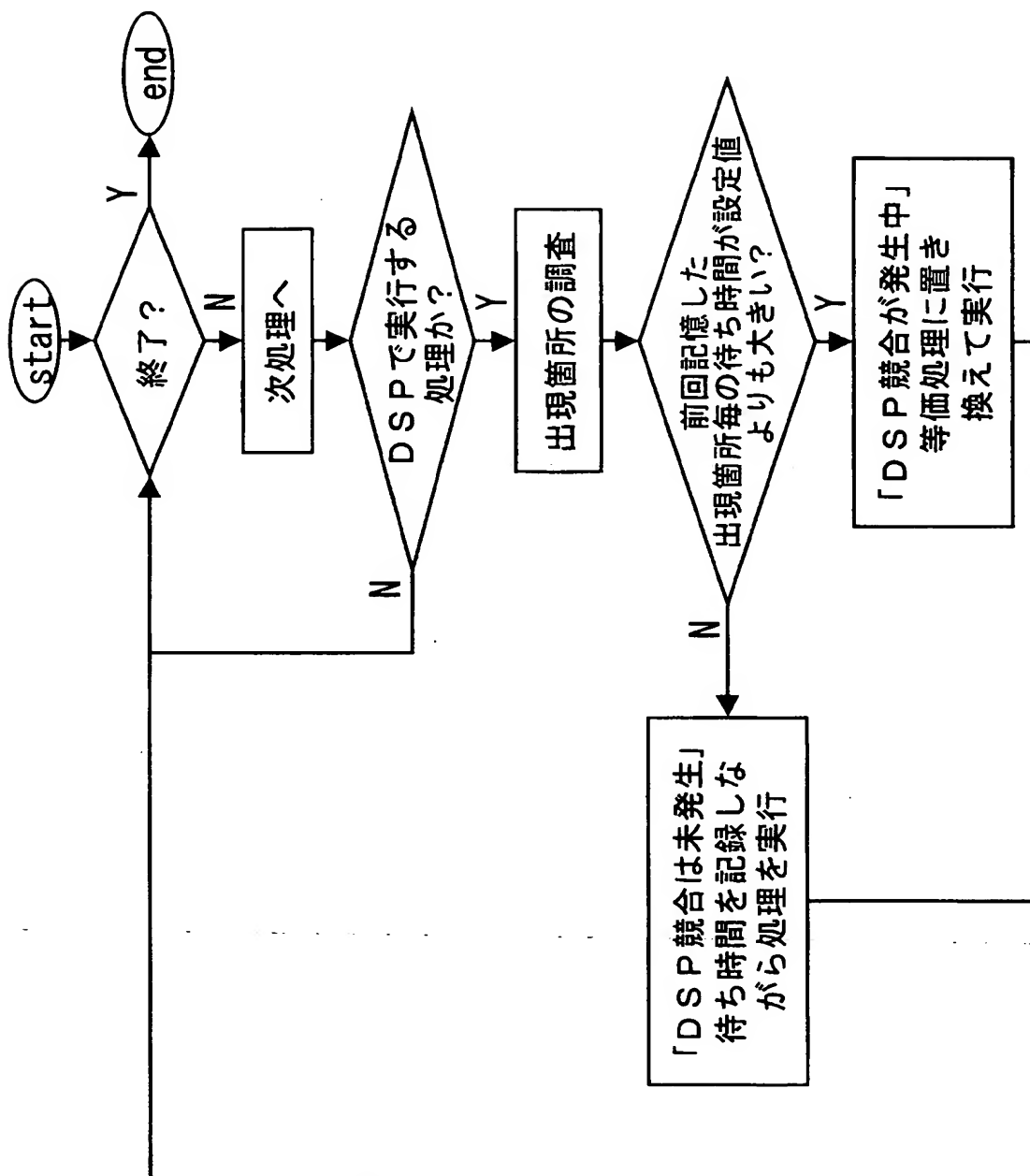
【図 25】



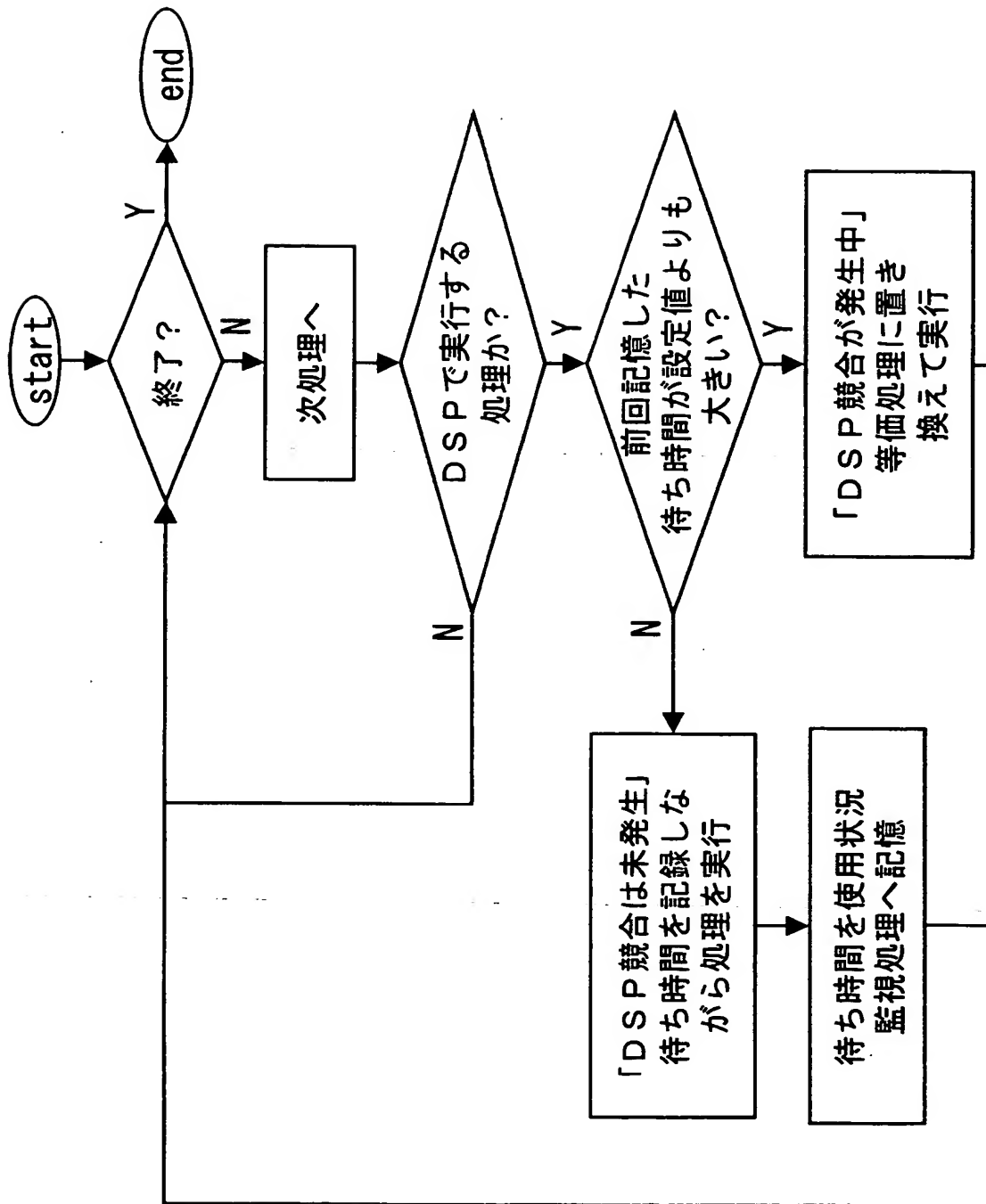
【図 2 6】

待ち時間 設定値	等価処理へ置き 換える確率[%]	等価処理へ置き 換えない確率[%]
~2. 0	60	40
2. 0~1. 8	70	30
1. 8~1. 6	80	20
1. 6~1. 4	90	10
1. 4~1. 2	95	5
1 以下	0	100

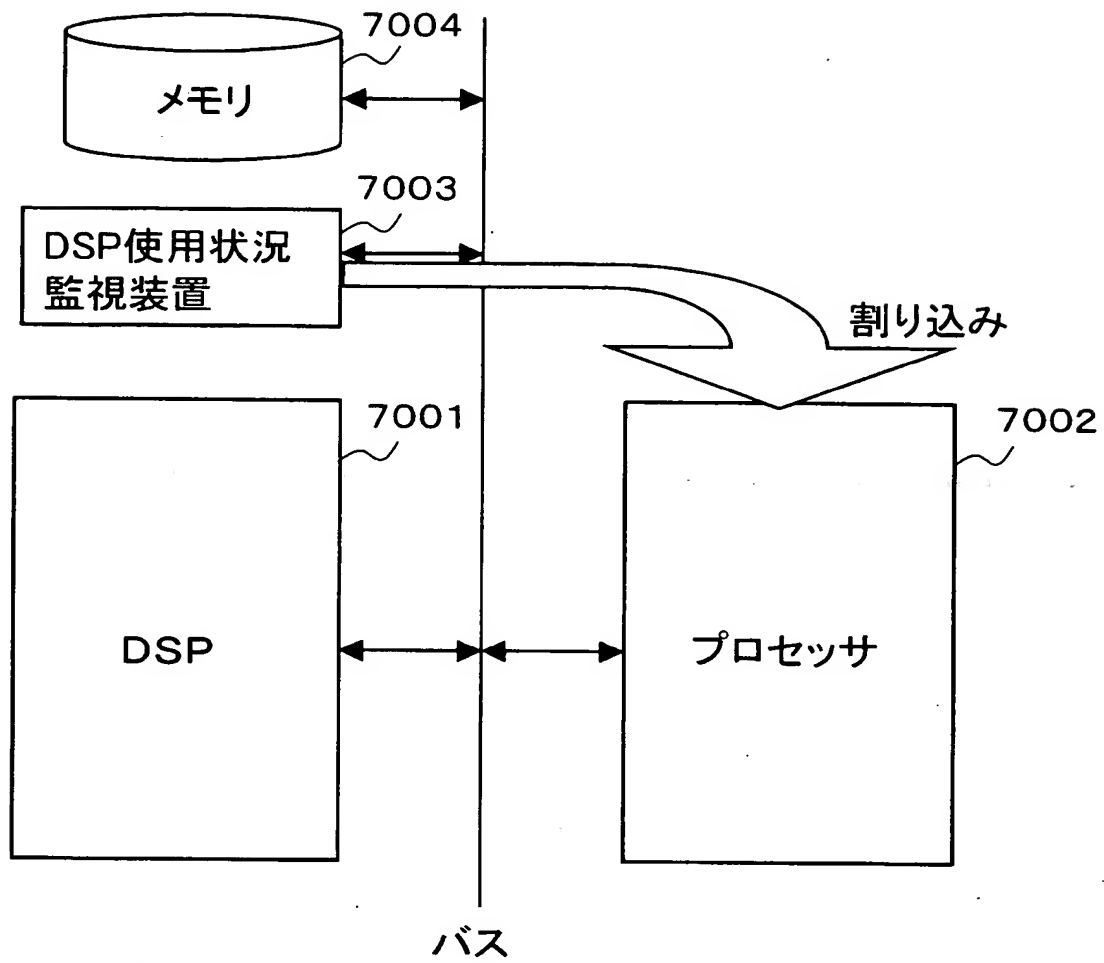
【図 27】



【図 28】



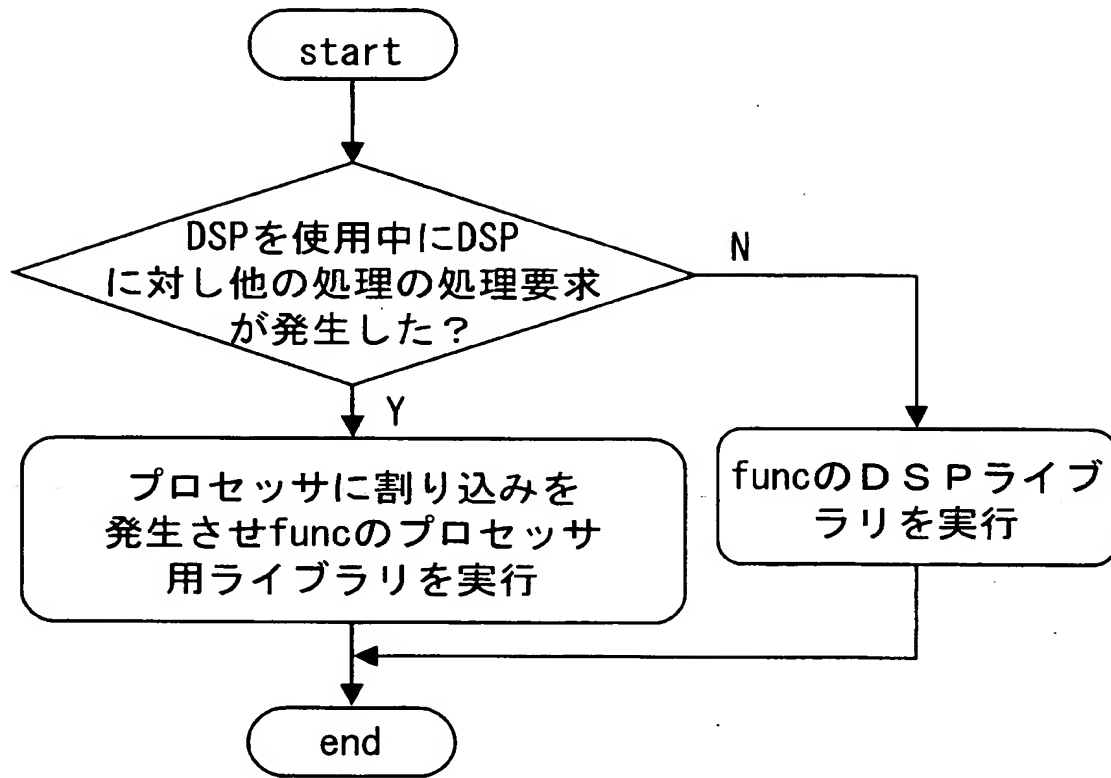
【図 29】



【図 30】

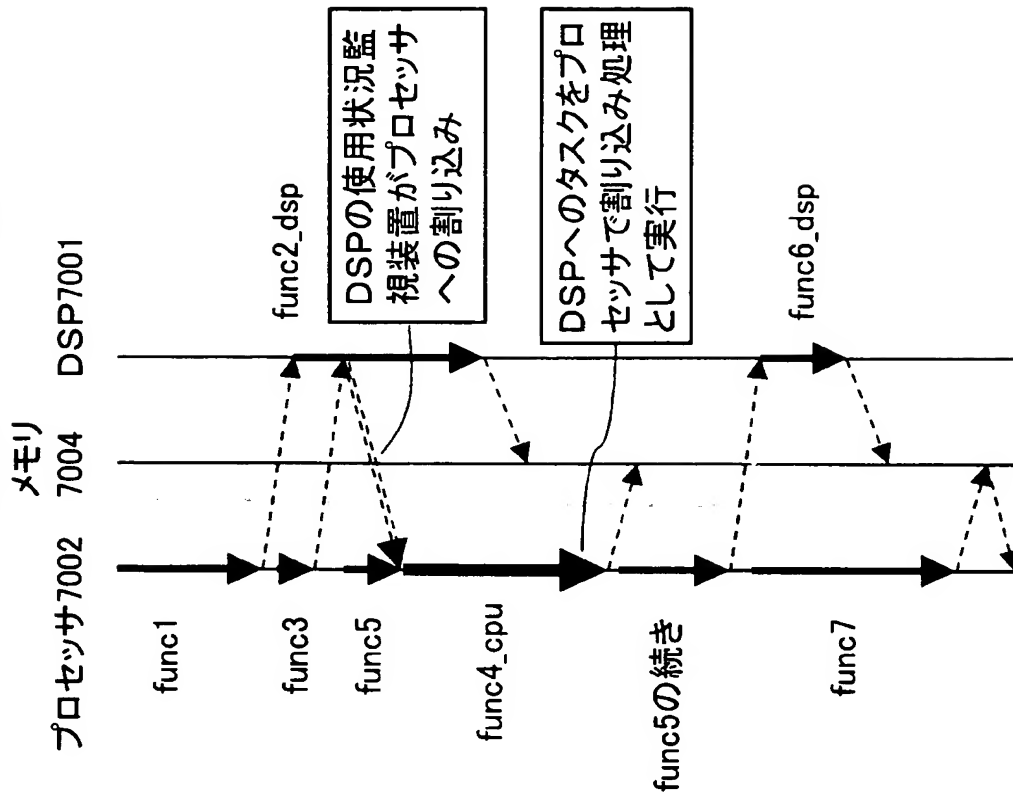
```
1:  main( )  
2:  {  
3:      :  
4:      func1()  
5:      func2()  
6:      func3()  
7:      func4()  
8:      func5()  
9:      func6()  
10:     func7()  
11:     func8()  
12:     :  
13: }
```


【図 31】

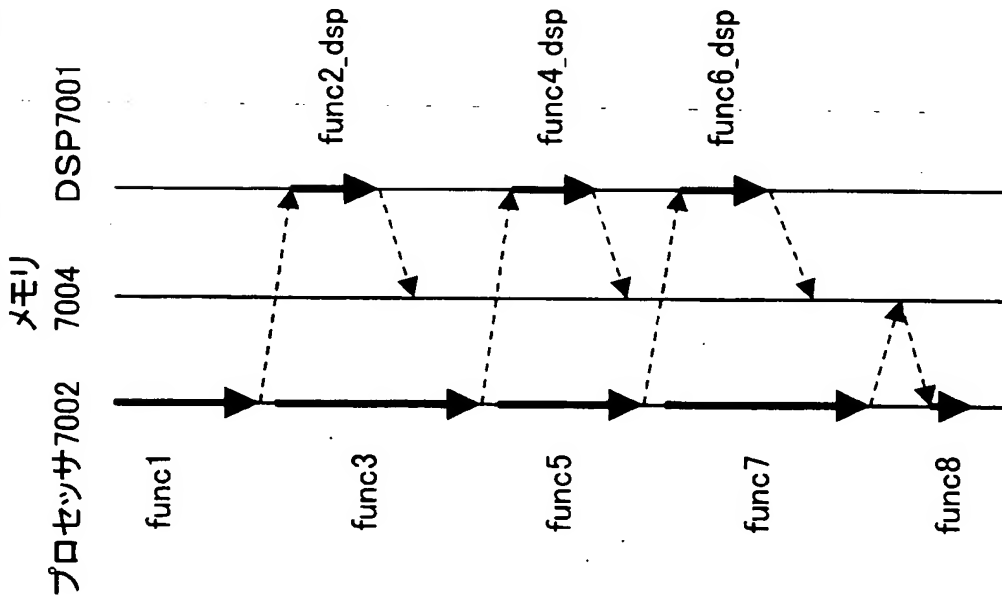


【図 3 2】

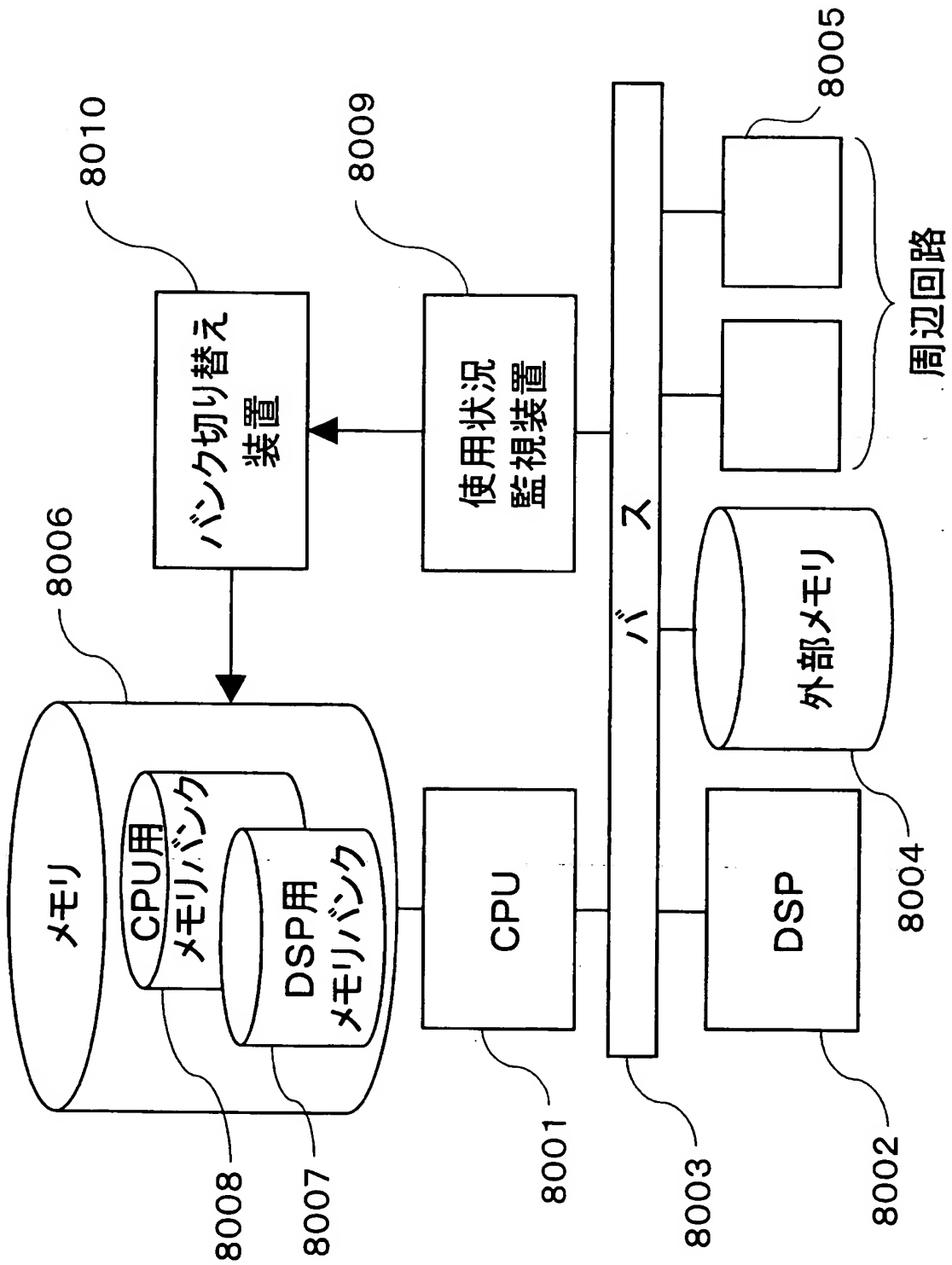
(b) DSPのタスクが競合した場合



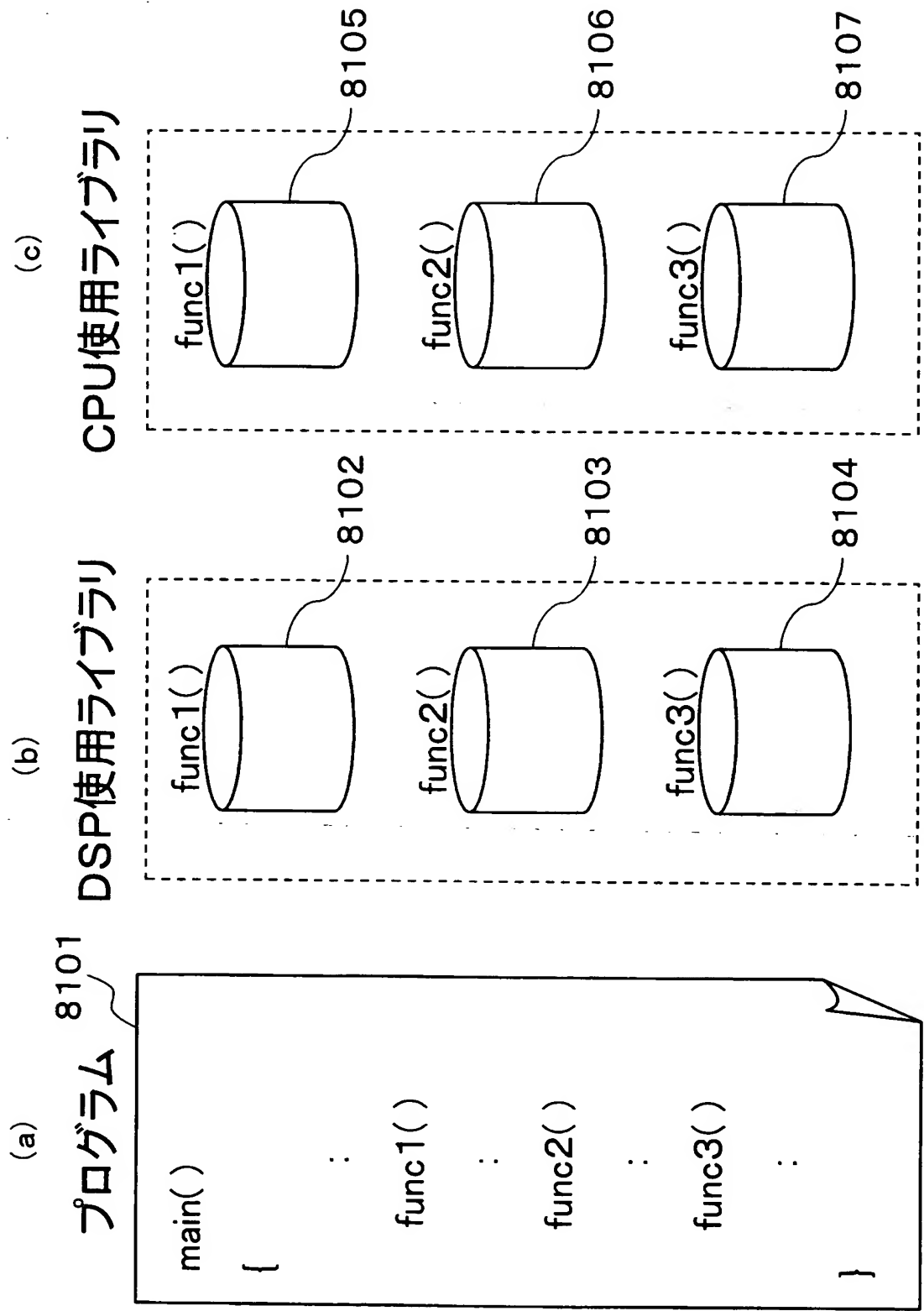
(a) DSPのタスクが競合しない場合



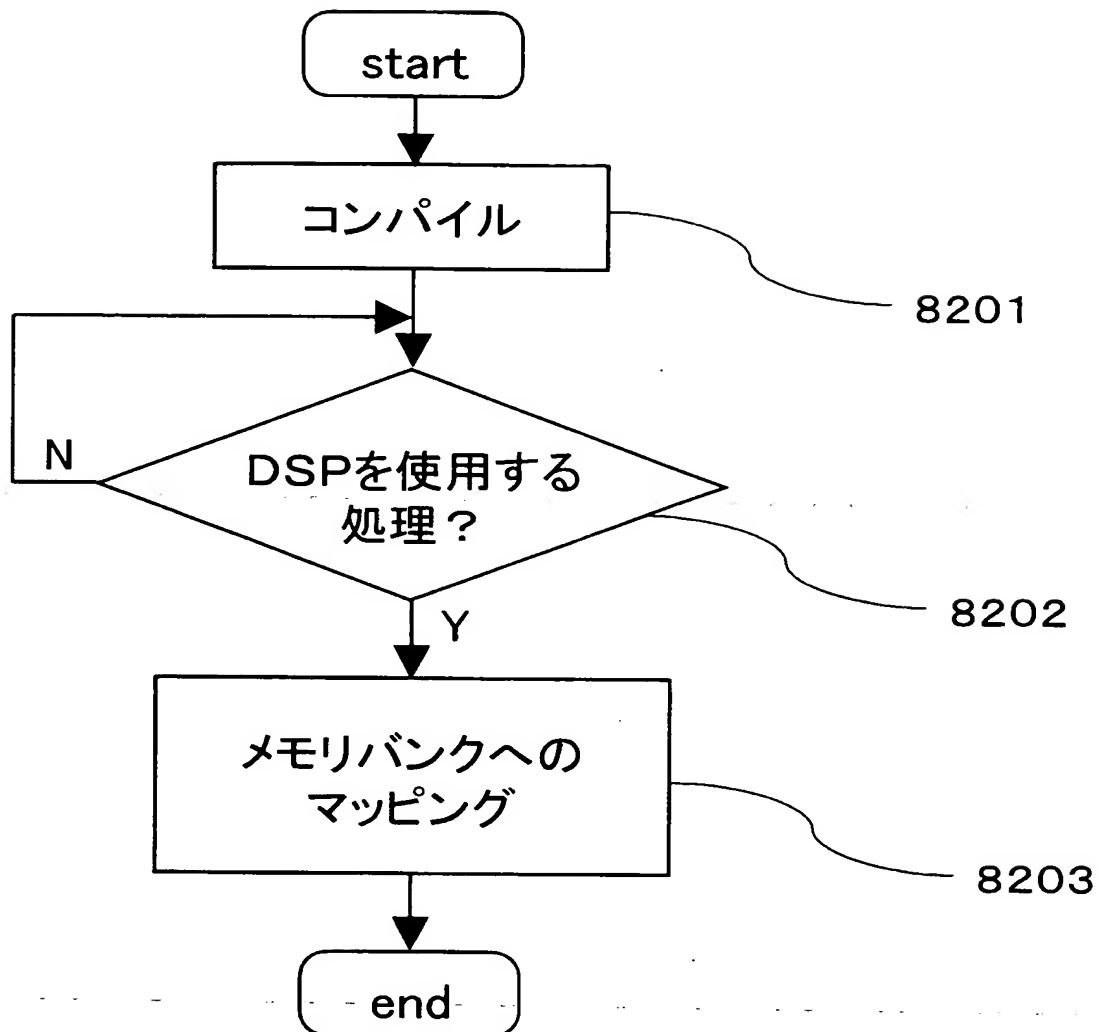
【図 33】



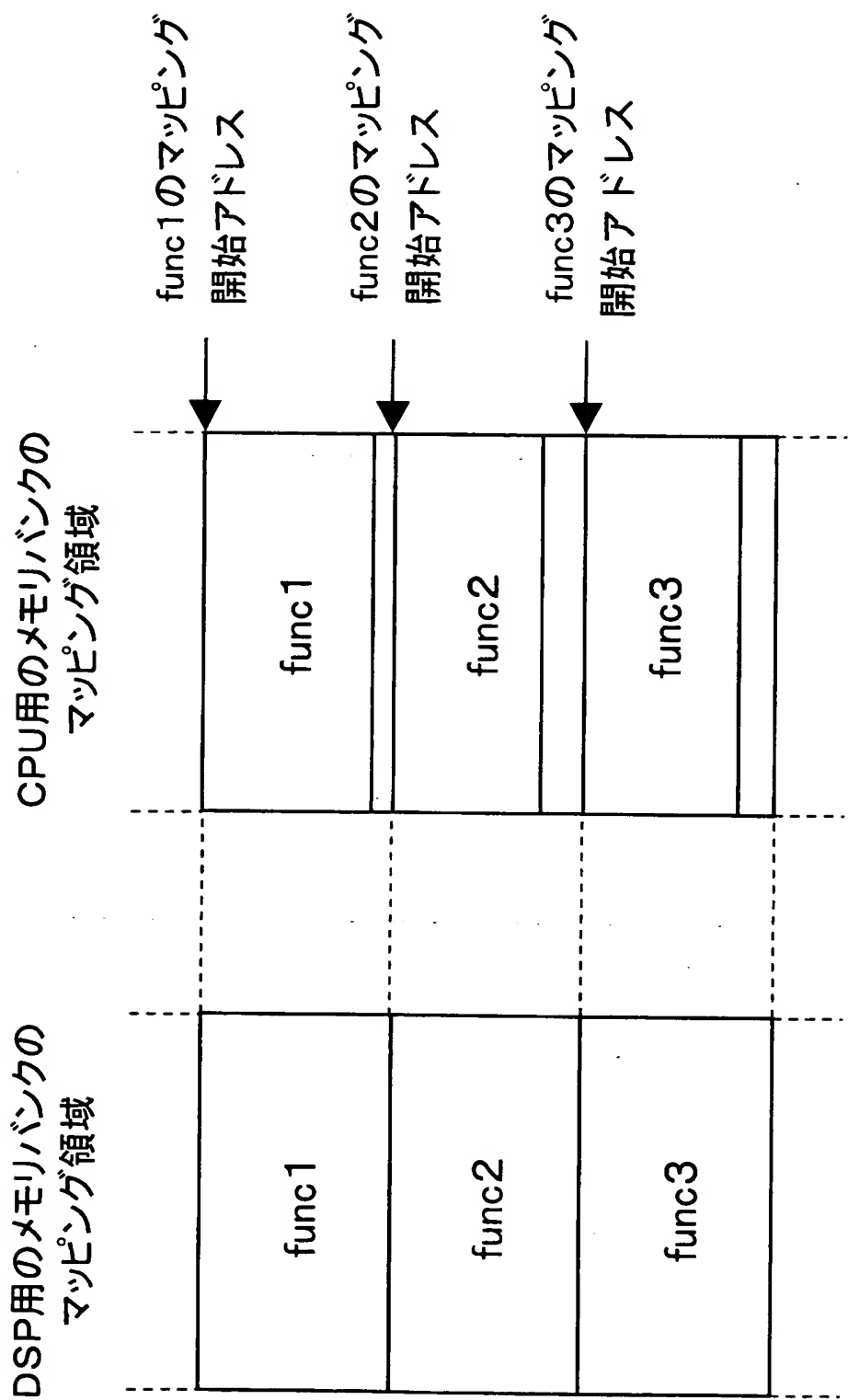
【図 34】



【図 35】



【図 36】



【書類名】 要約書

【要約】

【課題】 従来、バスやDSPなどの共有リソースを有するシステムにおいて、リソースを使用するソフトウェア処理が行われる際に、バスアクセス競合やDSP競合などのリソースの競合が頻繁に発生すると、結果としてシステム全体の性能が低下するという影響があった。本発明はかかる点に鑑み、リソースの競合の情報を考慮したソフトウェア処理システムを提供することを目的とする。

【解決手段】 リソースの使用状況を監視する使用状況監視処理の過程と、使用状況監視処理の過程の出力の情報に応じて実行されるソフトウェアの処理方法を変更するソフトウェア処理変更処理の過程を、ハードウェアまたはソフトウェアとして実現して、リソースの競合を発生しにくくしたものである。

【選択図】 図1

認定・付加情報

特許出願の番号	特願 2 0 0 2 - 3 5 5 6 8 3
受付番号	5 0 2 0 1 8 5 3 7 0 5
書類名	特許願
担当官	第七担当上席 0 0 9 6
作成日	平成 1 4 年 1 2 月 1 9 日

< 認定情報・付加情報 >

【提出日】 平成14年12月 6日

次頁無

特願 2002-355683

出願人履歴情報

識別番号

[000005821]

1. 変更年月日

1990年 8月28日

[変更理由]

新規登録

住 所

大阪府門真市大字門真1006番地

氏 名

松下電器産業株式会社